VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Microsoft Robotics Studio – Programování robotů

Microsoft Robotics Studio – Robots Programming

2010 Marián Poláček

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne, pričom som uviedol všetky literárne pramene a zdroje, z ktorých som čerpal.

V Ostrave 4. mája 2010 .....

Ďakujem všetkým, ktorí ma pri práci podporili. Menovite Ing. Tomášovi Kocyanovi, vedúcemu práce za cenné rady a Ing. Janovi Martinovičovi, Ph.D za náhľad na riešenie úloh.

Abstrakt

V posledných rokoch sa robotika stala dostupnou širšiemu okruhu užívateľov, čím bol umožnený jej prudký rozvoj. Sprístupnenie robotiky napríklad žiakom základných škôl bolo umožnené vývojom technológií, ktoré dovoľujú konštrukciu a programovanie robotov bez znalosti pokročilých programovacích techník. Zámerom tejto práce je ukážka spolupráce LEGO Mindstorms NXT a Microsoft Robotics Studio. Okrem toho sa práca zameriava na určovanie polohy robotov, pričom využíva knižnicu Emgu CV určenú pre prácu s grafikou. Súčasťou práce je popis problémov, ktoré boli pri vývoji nájdené a popis ich možného riešenia.

Kľúčové slová:

MRDS, Microsoft Robotics Studio, LEGO Mindstorms NXT, perspektívna projekcia, projektívna transformácia

Abstract

Robotics has become accesible to wider audience of users in last few years, what allowed to its rapid development. Making robotics accesible for example to students of primary schools was caused by development of new technologies, which allow construction and programming of robots without the requirement of advanced programming knowledge. The goal of this thesis is to show interaction between LEGO Mindstorms NXT and Microsoft Robotics Studio. In addition the thesis focuses on how to identify position of robots, using the Emgu CV library focused on image processing. The thesis also contains description of problems that were found in development and their possible solutions.

Keywords:

MRDS, Microsoft Robotics Studio, LEGO Mindstorms NXT, perspective projection, projective transformation

Zoznam použitých skratiek a symbolov

MRDS	-	Microsoft Robotics Studio
CCR	-	Concurrency and Coordination Runtime
DDS	-	Decentralized Software Services
VPL	-	Visual Programming Language
MSDN	-	Microsoft Developer Network
UI	-	User Interface
REST	-	Representational State Transfer
VSE	-	Virtual Simulation Environment
GPS	-	Global Posotioning System
URI	-	Unified Resource Identifier

Obsah

Zoznam obrázkov

Zoznam tabuliek

Výpisy z kódu

1 Úvod

Cieľom tejto práce je priblíženie vývojového prostredia Microsoft Robotics Studio a možností jeho využitia pri ovládaní robotického hardwaru reprezentovaného stavebnicou LEGO Mindstorms NXT. Cieľom práce tak je, okrem iného, aj návrh ovládania robotov. Za najdôležitejšiu časť práce sa považuje vytvorenie systému schopného lokalizovať jednotlivých robotov v rámci ihriska.

Táto práca spadá pod projekt fotbalu robotov, na ktorom sa zúčastňuje viacero študentov. Tento projekt sa snaží ponúkať riešenia pre ovládanie, identifikáciu a tvorbu stratégií hry robotov.

Druhá kapitola ponúka popis prostredia Microsoft Robotics Studio (MRDS), určeného pre tvorbu robotických aplikácií. V kapitole je uvedený popis vývojových nástrojov a niektorých kľúčových technológií, ktoré prostredie využíva.

Nasledujúca kapitola je venovaná konštrukcii robota využitého pre testovanie aplikácií. Robot je založený na stavebnici LEGO Mindstorms NXT, ktorej popis je taktiež súčasťou kapitoly.

Piata kapitola popisuje, akým spôsobom je možné využiť MRDS k programovaniu aplikácií pre LEGO Mindstorms NXT. Jednou zo súčastí kapitoly je aj implementácia vlastnej služby pre ovládanie robota.

Cieľom šiestej kapitoly je zodpovedanie otázky prečo, je potrebné robotov na ihrisku identifikovať. Okrem tohto je druhá časť kapitoly venovaná spôsobu identifikácie, ktorý bol použitý pri vytváraní aplikácie identifikujúcej robotov.

Ďalšia kapitola sa venuje teoretickým problémom, ktoré sú spojené so spracovaní obrazu zachyteného kamerou. V kapitole sú tak uvedené vysvetlené niektoré základné pojmy, ako napríklad homogénne súradnice.

Ôsma a zároveň posledná, kapitola popisuje algoritmy, ktoré boli použité pri tvorbe aplikácie identifikujúcej robotov v obraze. Aplikácia je schopná zo získaného obrazu určiť kód a polohu robotov na ihrisku, spolu s ich otočením.

2 Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio (MS RDS, MRDS) je prostredie pre simuláciu, kontrolu a vývoj robotov. MRDS je vyvíjané pre systém Windows a je založené na prostredí .NET. Je navrhnuté tak, aby dokázalo poskytnúť vývojové prostriedky, čo najširšej škále užívateľov od amatérov až po profesionálnych vývojárov. Vzhľadom na tento široký záber podporuje takmer akýkoľvek robotický hardware. Informácie použité v nasledujúcej kapitole boli čerpané z MSDN stránky, ktorá je venovaná tomuto projektu [1].

2.1 História a súčasnosť MRDS

Prvá verzia MRDS bola pod názvom Microsoft Robotics Studio vydaná v roku 2006. Obsahovala tri časti, ktoré s istými zmenami pretrvali až do súčasnej verzie. Išlo o vizuálny nástroj pre vytváranie robotických aplikácií, 3D simulačný nástroj a platformu pre správu a tvorbu služieb. Prvá verzia MRDS trpela hlavne nedostatkom príkladov a dokumentácie, napriek tomu začala okolo nej vznikať komunita vývojárov zaoberajúcich sa robotikou.

Vývojári MRDS sa okrem odstraňovania chýb a pridávania novej funkcionality zamerali aj na vytvorení ucelenej dokumentácie. V súčasnosti je táto dokumentácia súčasťou MSDN. MRDS tu má vlastnú sekciu, ktorá okrem popisu API obsahuje aj množstvo ukážok a tutoriálov. Oproti prvej verzii prebehla aj integrácia s prostredím Visual Studio 2005 a neskôr 2008. Vďaka tomu, už nie je nutné generovať veľkú časť projektu pomocou príkazového riadku.

2.2 Súčasti MRDS

Nasledujúca sekcia bude venovaná hlavným komponentám MRDS. Pri ich vývoji bolo prihliadané na špecifickosť vývoja robotických aplikácií. Tie vyžadujú jednak vysoký stupeň asynchrónnosti a súčasne musia byť schopné komunikovať s veľkým množstvom uzlov v sieti. Ďalším špecifikom MRDS je použitie vizuálneho programovacieho jazyka, ktorý uľahčuje programovanie jednoduchých úloh.

2.2.1 Concurrency and Coordination Runtime (CCR)

CCR je DLL knižnica využiteľná ľubovoľným programovacím jazykom použiteľným v prostredí .NET Framework. Bola vytvorená s cieľom riešiť problémy spojené s asynchrónnosťou a paralelizmom, ako aj umožniť riešenie čiastočných zlyhaní (Partial Failtures). Je vhodné ju použiť v prípade, keď komponenty aplikácie sú voľne viazané (Loosely Coupled). Takto môžu byť komponenty vyvíjané nezávisle na sebe a nemusia závisieť od vnútorneho stavu ostatných komponent.

Triedy, ktoré sú súčasťou CCR sa rozdeľujú do troch hlavných kategórií, podľa ich funkcionality.

- Triedy Port a PortSet sú generické triedy, ktorých cieľom je umožniť komunikáciu medzi službami. Port je implementáciou FIFO (First In First Out) rady,

ktorá združuje položky daného dátového typu. PortSet potom združuje skupinu Portov do kolekcie.

- Ďalšou kategóriou sú arbitre (Arbiters), triedy tohto druhu sú zodpovedné za koordináciu. Naväzujú sa na porty, poprípade portsety. Po naviazaní reagujú na nimi zasielané správy. Existuje viacero druhov arbitrov, ktoré sa vytvárajú pomocou statickej triedy Arbiter. Ako arbitre väčšinou vystupujú delegáti metód, ktoré sa potom vykonávajú po splnení určených podmienok. Pri vytváraní arbitra je možné špecifikovať, či má reagovať iba na prvú prijatú správu, alebo má obsluhovať všetky správy zasielané na port.

- Do poslednej kategórie patria triedy Dispatcher, DispatcherQueue a rozhranie ITask. Tieto triedy oddeľujú logiku plánovania (Scheduling) a rozdeľovanie záťaže (Load Balancing) od zbytku aplikácie. CCR nepoužíva štandardný thread pool, ktorý je implementovaný v CLR. Namiesto toho umožňuje vytvorenie izolovaných systémových thread poolov. Trieda Dispatcher v tomto systéme udržuje sadu vláken, ktorých počet je možné špecifikovať v konštruktore. DispatcherQueue funguje, ako FIFO fronta úloh a je jediným spôsobom komunikácie medzi softwarovými komponentami a Dispatcherom. Ako úlohy vystupujú triedy, ktoré implementujú rozhranie ITask. Príkladom takýchto tried sú napríklad všetky arbitre. Pre DispatcherQueue je možné špecifikovať jeho prioritu, takže po vytvorení viacerých front je zaistené, že kritické úlohy budú vykonané ako prvé. V praxi tak môžu až stovky komponent zdieľať jeden Dispatcher, ktorý rozdeľuje milióny úloh naprieč malým počtom systémových vláken.

2.2.2 Decentralized Software Services (DSS)

DSS je jednoduché (lightweight) prostredie založené na CCR. Poskytuje servisný model, ktorý kombinuje REST (Representational State Transfer) architektúru s prístupom k systému. Služby sú poskytované ako zdroje, ku ktorým je možné pristupovať z programu, alebo pomocou užívateľského rozhrania. Hlavným cieľom DSS je vytvorenie jednoduchej, ale výkonnej a robustnej platformy.

2.2.3 DSS Služby

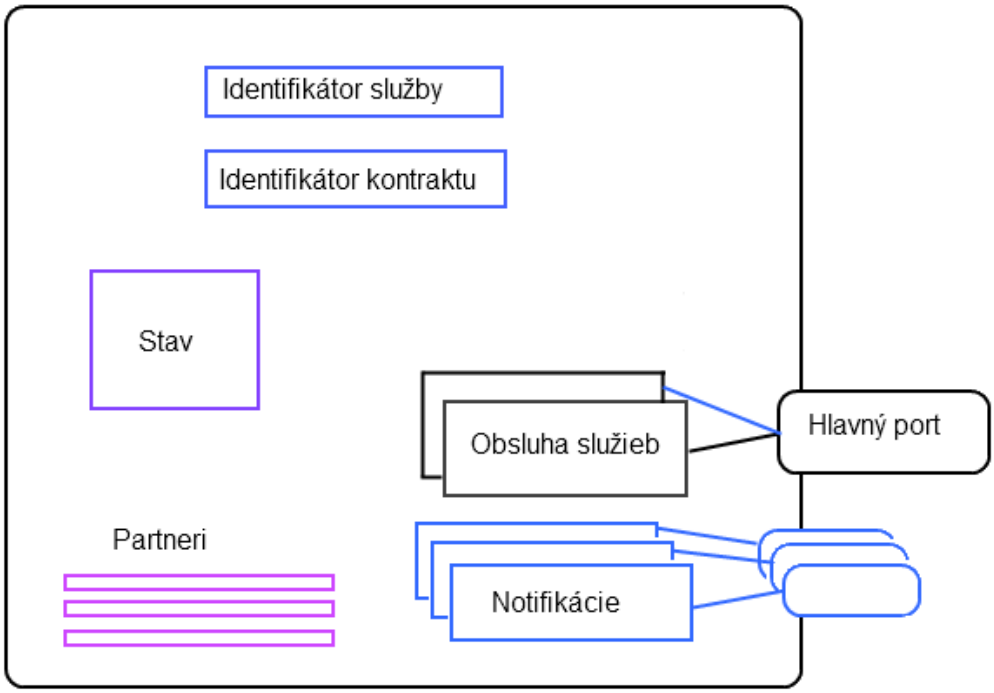
Služby slúžia ako základné stavebné bloky v aplikáciach vytvorených v MRDS. Pod pojmom služby si je možné predstaviť rozličné hardwarové (senzory, motor), alebo softwarové komponenty (napríklad UI komponenty).

Jednotlivé služby sa skladajú z nasledujúcich súčastí:

- Identifikátor služby - využíva sa jednoznačný identifikátor služby (URI). Ten je dynamicky vygenerovaný pri vytváraní služby. URI označuje špecifickú inštanciu

služby v rámci DSS uzla. Identifikátor umožňuje komunikáciu medzi službami a zároveň umožňuje prístup ku službe pomocou webového prehliadača.

- Identifikátor kontraktu - kontrakt je popis služby. Udáva, ako je služba implementovaná, jej správanie a ako sa dá využiť. Kontrakty sa používajú na generovanie DSS proxy knižníc, ktoré umožňujú prístup ku službe z kódu. Identifikátorom kontraktu je URI, ktoré jednoznačne identifikuje kontrakt. Identifikátor je automaticky generovaný po vytvorení projektu služby.



Obrázok 1: Model DSS služby

- Stav - stav je reprezentácia služby v určitom časovom okamžiku. Pod stavom si môžeme predstaviť dokument, ktorý popisue aktuálny obsah služby. Napríklad služba reprezentujúca klávesnicu, môže obsahovať informáciu o práve stlačených klávesoch. Ak je potrebné, aby bolo možné istú informáciu sledovať, získať alebo upravovať, je nutné ju zahrnúť do stavu služby. Okrem zachytávania obsahu služby je možné stav využiť na riadenie užívateľského rozhrania služby.

- Partneri - partnermi sú ďalšie služby, s ktorými služba komunikuje, alebo je na nich závislá. Vytvorenie zoznamu partnerských služieb oznamuje prostrediu DSS Node, s ktorými službami sa ma nadviazať spojenie. Pri deklarácii partnerov je možné špecifikovať typ vzťahu medzi službou a partnerom. Ak ide o vzťah závislosti, služba

nemôže byť vytvorená, bez vytvorenia spojenia s partnerom. Pri nepovinnom vzťahu je služba vytvorená bez ohľadu na spojenie.

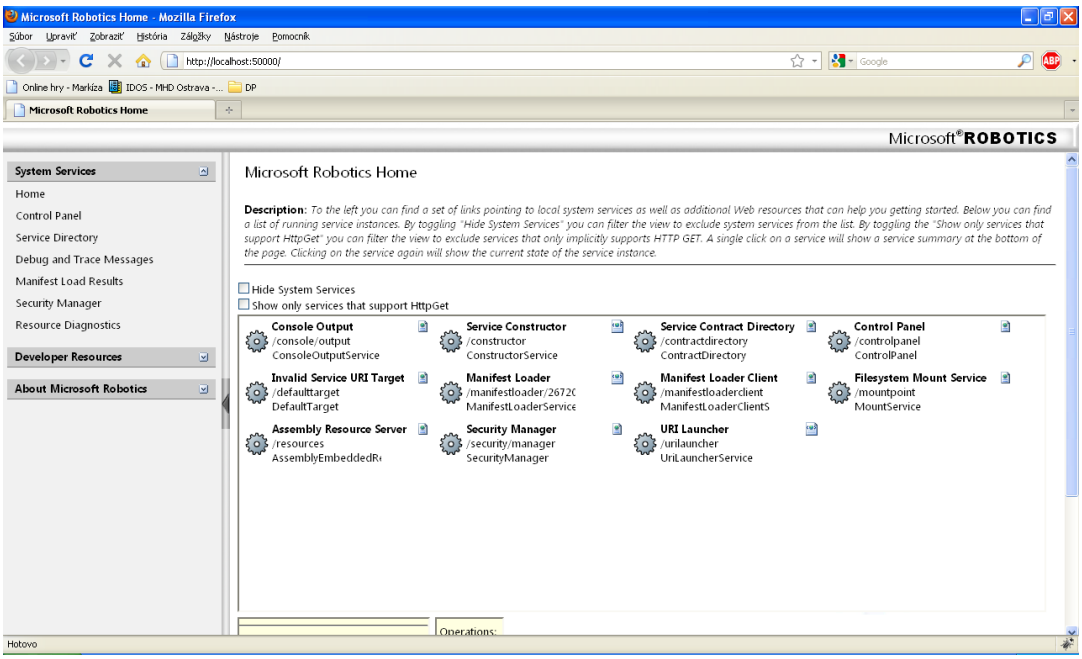
- Hlavný port (Main Port) - ide o CCR Port, ktorým prichádzajú správy od ostatných služieb. Služby medzi sebou dokážu komunikovať iba pomocou zasielania správ na hlavný port. Typ správ, ktoré port akceptuje je definovaný typom portu.

- Obsluha služieb (Service handlers) - pre každú operáciu, ktorá je definovaná pre hlavný port, musí byť definovaná obslužná metóda. Jedinou výnimkou sú operácie DsspDefaultLookup a DsspDefaultDrop, pre ktoré existujú štandardné implementácie.

- Notifikácie (Event Notifications) - pri zmene svojho stavu služba obvykle vygeneruje notifikácie. Na prijatie notifikácie o zmene stavu inej služby sa u nej musí služba zaregistrovať. Potom môže prijímať notifikácie pomocou samostatných CCR portov.

2.2.4 DSS Uzly

DSS uzol (DSS node) je kontext, v ktorom sa vykonávajú služby. Pre vytvorenie DSS uzla je možné použiť aplikáciu dsshost.exe, alebo je možné využiť odkaz v hlavnom menu. Ak je DSS uzol vytvorený bez parametrov, aktivujú sa na ňom iba služby nutné pre vytváranie ďalších služieb. Služba vytvorená v kontexte uzlu potom pracuje, až do svojej deaktivácie,

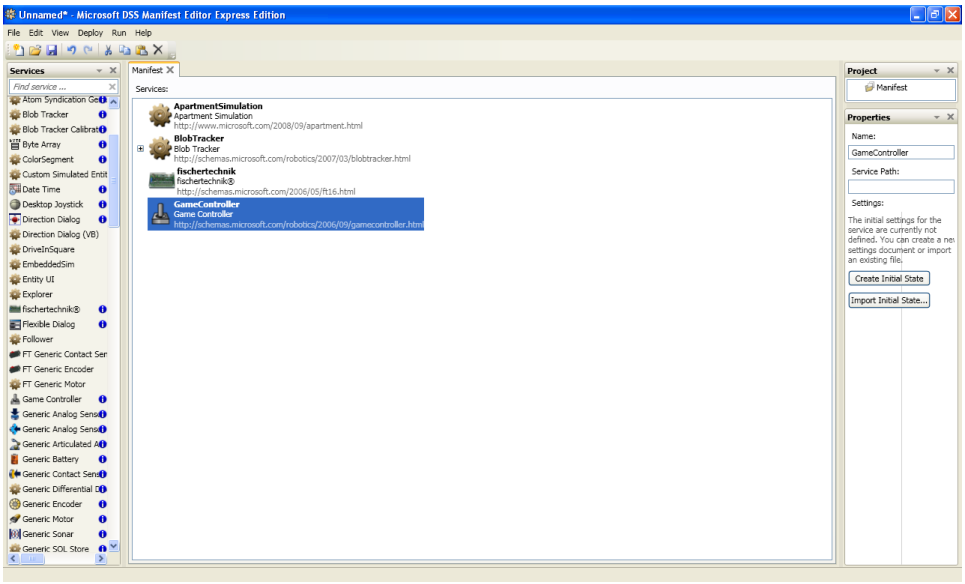


Obrázok 2: Webové rozhranie DSS uzlu

alebo do okamihu vypnutia DSS uzlu.

DSS uzol poskytuje užívateľské rozhranie dostupné z webového prehliadača. Defaultná adresa, na ktorej je možné uzol nájsť je http:\\localhost:50000. Pri vytváraní uzlu je možné zmeniť port, na ktorom má byť uzol dostupný.

2.2.5 Manifesty



Obrázok 3: DSS Manifest Editor

Pre konfiguráciu služieb, ktoré sa majú aktivovať v prostredí DSS uzlu sa využívajú XML súbory nazvané manifesty. Manifest predstavuje množinu služieb, ktorá sa má aktivovať na danom DSS uzle.

Po aktivácií uzla sa na ňom automaticky spustí služba nazvaná Manifest Loader Service. Tá je zodpovedná za spracovanie manifestov a musí byť aktívna počas celého života uzla. Manifesty môžu byť načítané v ľubovoľný okamih, ale väčšinou sa využíva načítanie manifestov pri spustení uzla. V takom prípade sa manifest predáva ako jeden z parametrov spúšťacej aplikácií.

Pred spracovaním službou Manifest Loader Service je telo manifestu validované voči jeho XML schéme. Telo manifestu tvorí zoznam záznamov popisujúcich jednotlivé služby. Záznamy môžu obsahovať parametry popisujúce stav služby. Tiež je možné vložiť do záznamu odkaz na XML súbor popisujúci jej plný stav.

Po prečítaní manifestu predáva Manifest Loader Service získané informácie ďalšej zo základných služieb nazvanej Constructor Service. Táto služba vytvára inštancie služieb volaním ich konštruktora. Ako parametre sa použijú získané záznamy. Vďaka takémuto rozdeleniu je možné vytvárať služby aj pomocou priameho volania Constructor Service, bez nutnosti špecifikovať manifest.

Pre vytváranie a editáciu manifestov je možné použiť ľubovoľný textový editor. MRDS však obsahuje aj nástroj, ktorý sa špecializuje na úpravu manifestov. Ide o DSS Manifest Editor. S jeho pomocou je možné vytvárať manifesty v grafickom prostredí, čo môže byť v niektorych prípadoch prehľadnejšie, ako manuálna editácia XML súborov.

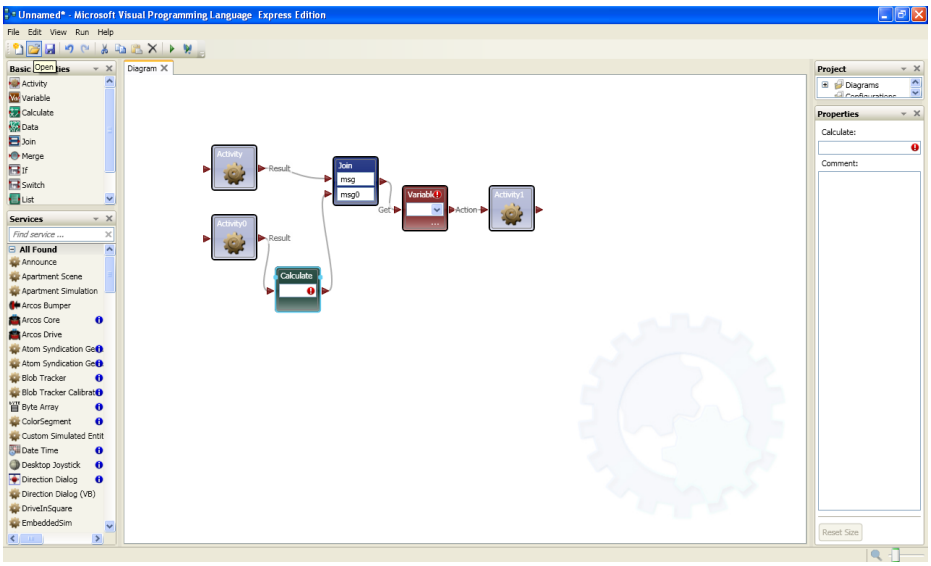
Hlavnou výhodou využitia manifestov je v možnosti deklarovať služby oddelene od ostatného kódu. Takto je umožnené napríklad jednoducho prejsť od použitia simulovaného hardwaru k použitiu reálnych komponent iba zmenou manifestu.

```
<?xml version="1.0" ?>
<Manifest
   xmlns="http://schemas.microsoft.com/xw/2004/10/manifest.html"
   xmlns:dssp="http://schemas.microsoft.com/xw/2004/10/dssp.html" >
   <CreateServiceList>
      <ServiceRecordType>
         <dssp:Contract>
                  http://schemas.tempuri.org/2010/05/robotcontrolservice.html
          </dssp:Contract>
      </ServiceRecordType>
   </CreateServiceList>
</Manifest>
```

Výpis 1: Ukážka základného manifestu

2.2.6 Vizuálny programovací jazyk

Visual Programming Language (VPL) je vývojové prostredie založené na grafickom programovacom modeli, ktorý zachycuje tok dát. Oproti obdobným prístupom,



Obrázok 4: Prostredie VPL

kde sa príkazy vykonávajú sekvenčne, vykonávajú komponenty VPL definované úlohy, iba ak obdržia na vstupe nejaké dáta. Vďaka tomu je možné riešiť rôzne úlohy založené na paralelizme, alebo úlohy, ktoré vyžadujú distribuované spracovanie.

VPL je cielené na začínajúcich programátorov, ktorí majú základné znalosti o premenných a logike programov. Neznamená to však, že by VPL nebolo využiteľné aj pokročilými užívateľmi. Vďaka možnosti generovania zdrojového kódu z VPL diagramu sa dá využiť napríklad na rýchle vytvorenie prototypu aplikácie. Po vygenerovaní kódu je potom možné pridať pokročilejšiu funkcionalitu tradičnejším spôsobom.

Tok dát je vo VPL reprezentovaný prepojenými blokmi aktivít. Prepojenia medzi aktivitami reprezentujú správy, ktoré putujú medzi aktivitami. Konkrétne aktivity sú modelované obdĺžnikmi obsahujúcimi názvy aktivít. Pod aktivitami si môžme predstaviť napríklad služby DSS, metódy toku dát, alebo ďalšie štruktúry definované v jazyky VPL. Aktivitu môže tvoriť aj súbor iných aktivít. Vďaka tomu je možné konštruovať znovupoužiteľné bloky (obdoba metód).

Vstupy aktivity sú umiestnené na jej ľavom okraji. Pre vstupy sú definované obslužné metódy (akcie alebo handlery), ktoré reagujú na určitý typ správ. Handlery generujú výsledky na výstupoch, ktoré sú umiestnené v pravom okraji aktivity. Výstup jednej aktivity môže samozrejme slúžiť ako vstup pre ďalšiu aktivitu.

2.2.7 Simulačné prostredie

Visual Simulation Environment (VSE) vzniklo, kvôli urýchleniu vývoja robotických aplikácií, a ako pokus rozšíriť a zjednodušiť vývoj robotov. Umožňuje simuláciu rozličných scenárov, ktoré si vyžadujú presnosť zobrazenia a rozšíriteľnosť. S využitím simulácie je možné navrhnúť robota, meniť jeho súčasti a testovať ho bez nutnosti vlastniť aktuálny hardware.

VSE využíva poznatky o 3D zobrazovaní získané z herného priemyslu. Pre vytváranie fyzikálnych modelov sa používa technológia PhysX vyvíjaná firmou AGEIA. PhysX umožňuje využitie fyzikálnych čipov, ktoré sú súčasťou niektorých grafických kariet. Ak nie je špecializovaný čip dostupný, použije sa emulácia pomocou CPU. Pre renderovanie scien sa využíva framework Microsoft XNA.

Využitie smulácie má niekoľko výhod a súčasne obmedzení, s ktorými je potrebné počítať ešte pred využitím simulácie v reálnom projekte.

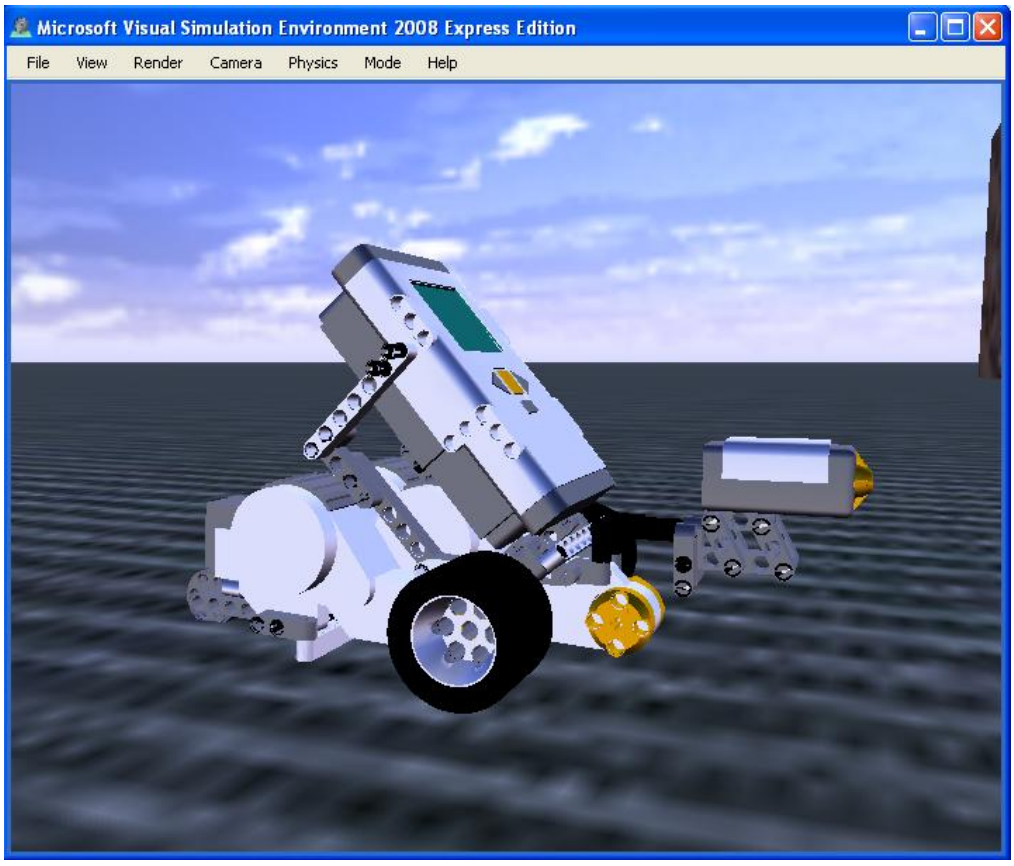
- Jednoduché použitie – užívateľské rozhranie simulačného prostredia je veľmi prehľadné, takže sa v ňom užívatelia dokážu rýchlo zorientovať.

- Prototypovanie – simuláciu je možné využiť pre tvorbu prototypov, bez nutnosti vlastniť skutočný hardware.

- Možnosť zdieľať robotov – táto výhoda vychádza z možnosti prototypovania. S virtuálnym robotom môže pracovať väčší počet ľudí, pričom nehrozí jeho poškodenie spôsobené nesprávnym naprogramovaním.

- Modifikovateľnosť – simulačné prostredie poskytuje možnosť úpravy (nielen) fyzikálneho modelu. Je tak možné simuláciu viac priblížiť reálnemu svetu. Inou možnosťou je simulácia extrémnych podmienok (napríklad stav beztiaže, extrémny tlak).

- Simulácia náhodných javov – simulovať rôzne náhodné javy reálneho sveta je v simulačnom prostredí veľmi obtiažne.



Obrázok 5: Simulačné prostredie

- Dolaďovanie simulácie – vytvorenie základnej simulácie je pomerne jednoduché. Problémom býva vyladenie simulovaného hardwaru, aby čo najviac zodpovedal skutočnému svetu. Tento problém býva často spôsobený neúplnou (alebo nepresnou) dokumentáciou dodávanou k hardwarovým komponentám.

- Nepresnosť – aj keď simulačné prostredie poskytoje pomerne dobré informácie o správaní simulovaných objektov, nie je možné simulovať úplne všetky javy reálneho sveta. Tento problém sa môže prejaviť hlavne v neskorších fázach projektov, keď dáta získané simuláciou nie sú dostatočné. Z tohto dôvodu je výhodné simuláciu využívať v pri tvorbe prototypu a na základné otestovanie správania hardwaru. Po určitom čase je potrebné prejsť na testovanie v reálnom svete, ktoré môže odhaliť ďalšie problémy, poprípade poskytnúť presnejšie informácie.

2.3 Zhodnotenie MRDS

Microsoft Robotics Developer Studio predstavuje zaujímavý koncept. Na jednej strane je prístupné začínajúcim nadšencom robotiky, ale obsahuje dostatok možností, aby uspokojilo potreby aj zložitejších aplikácií. Súčasti MRDS sú navrhnuté tak, aby boli použiteľné aj v iných oblastiach, ako v robotike. CCR spolu s DSS sú dostupné aj samostatne, pod názvom CCR and DSS Toolkit. Tento balík sa využíva sa v komerčných aplikáciách, ktoré musia riešiť paralelizmus a distribuované prostredie. Koncept VSE je použiteľný na testovanie robotov, ako aj na učenie sa základom robotiky bez nutnosti nakupovať hardware. VPL jazyk poskytuje jednoduchý nástroj pre tvorbu prototypov pre profesionálov a je ideálny pre menej zdatných programátorov.

3 Konštrukcia robota

Nasledujúca kapitola je venovaná popisu robota využitého pri testovaní aplikácie. Samotný návrh a konštrukcia robota neboli súčasťou tejto diplomovej práce. Robot bol vytvorený Jánom Pastrňákom v práci, na ktorú táto naväzuje. Cieľom tejto kapitoly preto nie je poskytnúť detailný postup konštrukcie robota, ale zhrnutie technológií použitých pri jeho tvorbe. Zdrojom informácií o stavebnici LEGO Mindstorms NXT bola stránka firmy LEGO venovaná tejto stavebnici [2]. Popis parametrov robota a niektoré informácie o jeho konštrukcii boli čerpané z práce Jána Pastrňáka [3].

3.1 LEGO Mindstorms NXT

Pri konštrukcii robota bola použitá stavebnica LEGO Mindstorms NXT. Ide o programovateľnú sadu pre vytváranie robotov vyvíjanú firmou LEGO.

Stavebnica bola vydaná v roku 2006, ako druhá generácia LEGO Mindstorms stavebníc. Prvá generácia mala prívlastok RCX. Verzia RCX využívala klasické LEGO kocky, zatiaľ, čo druhá generácia používa kocky LEGO Technics. Tie sú vhodnejšie na stavbu robotov, pretože umožňujú väčšiu voľnosť pri návrhu robota.

Firma LEGO na svojej stránke [2] poskytuje kompletnú dokumentáciu stavebnice a jej programovateľných súčastí. Okrem toho sú tam zverejňované zaujímavé konštrukcie a návody na programovanie takýchto robotov. Zaujímavým počinom sú aj rôzne súťaže zamerané hlavne na základné a stredné školy, ktorých účelom je spropagovať robotiku medzi širšou verejnosťou.

3.1.2 Súčasti stavebnice

Stavebnica obsahuje sadu kociek LEGO Technics, ktoré sa využívajú pre stavbu tela robota. Okrem nich existuje niekoľko programovateľných súčastí, ktoré sú základom pre interaktívnu funkčnosť robota. Jednotlivé súčasti sú popísané nižšie:

- NXT Intelligent Brick - Ide o riadiacu jednotku stavebnice. Je na nej umiestnený monochromatický LCD displej spolu so štyrmi ovládacimi tlačidlami. Okrem toho umožňuje prehrávať zvuky. Ako napájací zdroj je použitých šesť AA batérií, alebo je možné využiť Li-Ion batériu, ktorú je možné dobíjať bez vypnutia robota. Riadiaca jednotka umožňuje pripojenie štyroch vstupných zariadení (senzorov) a troch výstupných zariadení (motory). V Tabuľke 1 sú uvedené parametre jednotky.

- Pohonná jednotka - Pohonnú jednotku dodávanú so stavebnicou tvorí servomotor s integrovaným otáčkomerom. Otáčkomer dokáže merať otáčky motora v stupňoch, alebo v celých otáčkach a to s presnosťou jedného stupňa. Získané informácie dokáže odovzdávať riadiacej jednotke.

- Ultrazvukový senzor - Tento senzor umožňuje detekciu objektov. Okrem toho dokáže zmerať vzdialenosť od najbližšieho objektu v centimetroch. Maximálna zmerateľná vzdialenosť je 255cm, pri takejto vzdialenosti je odchýlka približne 3cm. Senzor funguje na princípe sonaru. Vysiela ultrazvukový signál do okolia a počíta čas od vyslania signálu po zachytenie odrazeného signálu detektorom. To znamená, že senzor dosahuje najlepšie výsledky pri detekcií predmetov pravidelného tvaru. Nepravidelné, poprípade malé, predmety môžu signál odraziť do všetkých strán, čo znemožňuje detekciu. Pri stavbe robota treba počítať s tým, že nie je možné využiť viacej, ako jeden takýto senzor, pretože viaceré senzory sa budú navzájom rušiť.

- Zvukový senzor - Senzor meria intenzitu zvuku v okolí. Dokáže zachytiť frekvencie medzi 3-6 kHz (sú to frekvencie, na ktoré je ľudský sluch najcitlivejší)

- Optický senzor - Nejedná sa o kameru v pravom zmysle slova. Optický senzor dokáže prijímať intenzitu svetla, takže robot dokáže rozlíšiť medzi svetlom a tmou. Okrem toho senzor obsahuje červenú diódu, pomocou ktorej dokáže určovať farby (analyzuje odtiene šedej, ktoré sa odrazia od farebnej plochy)

- Dotykový senzor - Skladá sa z tlačidla, ktoré dokáže reagovať na jedno, alebo viacnásobné stlačenie.

Procesor	Atmel ARM 7, 32-bit
RAM	64 KB
Pamäť	256 KB
USB	1x USB 2.0
Počet vstupných portov	4
Počet výstupných portov	3
Napájanie	6x AA batéria (každá 1.5V)
Iné príslušenstvo	Bluetooth

Tabuľka 1: Parametre NXT Intelligent Brick

3.1.3 Rozširujúce časti stavebnice

Vďaka tomu, že LEGO uvoľnilo kompletnú hardwarovú aj softwarovú dokumentáciu všetkých častí stavebnice, je možné vytvárať komponenty schopné spolupracovať s Mindstorms NXT. Dokonca existuje viacero firiem špecializujúcich sa na vytváranie príslušenstva pre túto stavebnicu (firmy ako HiTechnics, MindSensors). Vo väčšine prípadov sa jedná o rôzne dokonalé sensory, ktoré dodávajú práci so stavebnicou nové rozmery.

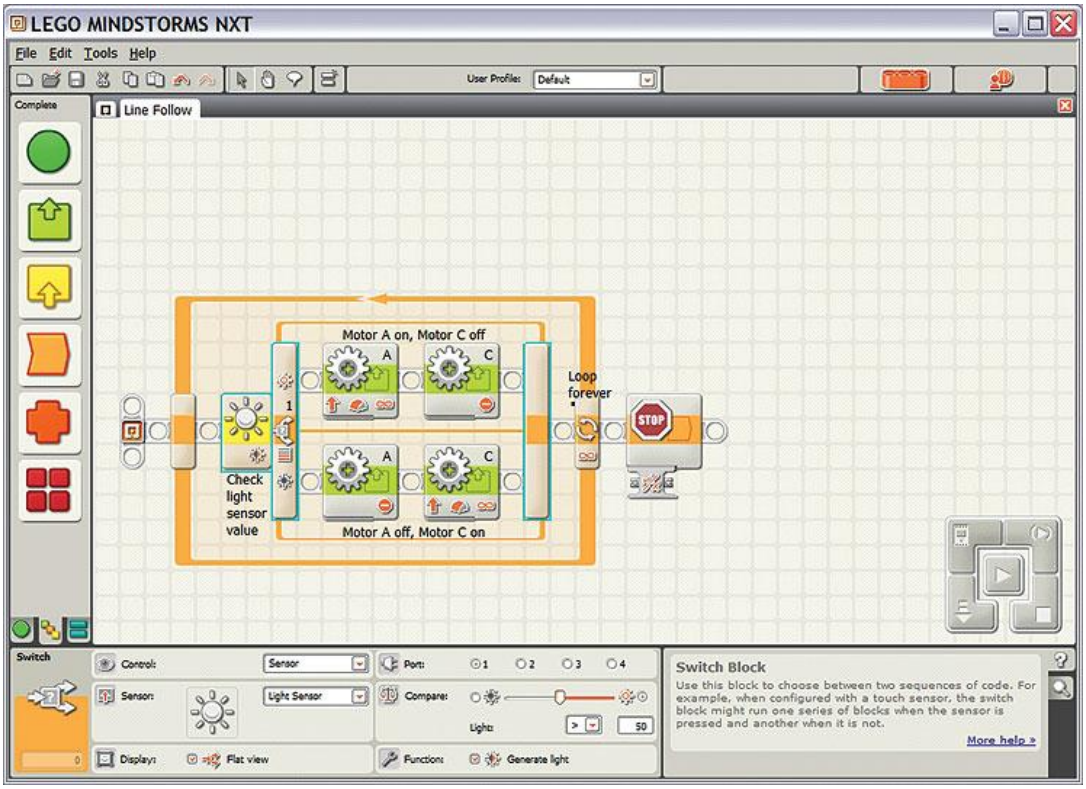
Na internete sa objavujú aj návody na konštrukciu vlastných doplnkov, takže zdatnejší užívatelia nie sú odkázaní na nákup component.

3.1.4 Tvorba softwaru pre Mindstorms NXT

Najzákladnejšie programy sa dajú vytvoriť pomocou menu, ktoré je súčasťou riadiacej jednotky. Komplikovanejšie programy je nutné na riadiacu jednotku nahrať pomocou USB portu, alebo Bluetooth.

Základný nástroj pre tvorbu programov pre Mindstorms NXT bol vyvinutý firmou LEGO. Je založený na prostredí LabVIEW, ktorý vyvíja National Instruments. Jedná sa o grafické programovacie prostredie. Programy sa vytvárajú zoraďovaním blokov do požadovanej konštrukcie. Bloky môžu reprezentovať jednotlivé hardwarové časti robota, alebo sofwarové konštrukcie (matematické operácie, porovnávanie, iterácie). Nástroj bol pomenovaný NXT-G a je dodávaný priamo so stavebnicou.

Vzhľadom pripomína prostredie NXT-G VPL, oproti ktorému je však obmedzené iba na programovanie pre stavebnicu Mindstorms NXT. Jeho výhodou je, že umožňuje jednoduchý prenos programu z počítača, na ktorom bol napísaný na NXT Brick pomocou USB alebo Bluetooth. Nevýhodou je veľkosť programov vytvorených v tomto prostredí. Oproti obdobnému programu napísanému v napríklad C++ môže mať program vytvorený v NXT-G až šesťnásobnú veľkosť. Pri komplikovanejších aplikáciách hrozí, že program bude väčší, ako kapacita NXT riadiacej jednotky.



Obrázok 6: Prostredie NXT-G

Okrem základného prostredia je možné využiť rozličné programovacie nástroje dodávané tretími stranami. Vo väčšine prípadov sa jedná o open-source projekty. Tieto nástroje môžu byť založené na jednom z populárnych jazykov (C++, C#, Java, Perl), alebo ponúkajú vlastné grafické prostredie (podobne ako NXT-G alebo VPL).

Programovacie prostredia sa dajú rozdeliť do dvoch skupín podľa toho, kde sa bude výsledný program vykonávať:

- Program sa vykonáva na riadiacej jednotke - do tejto skupiny patrí aj základný nástroj NXT-G. Program je po nahratí do riadiacej jednotky možné spustiť. Tento spôsob je, vzhľadom na obmedzený výkon, riadiacej jednotky vhodný pre menej náročné aplikácie. Robot sa potom na základe programu autonómne rozhoduje. Roboti môžu medzi sebou komunikovať napríklad pomocou Bluetooth, čo umožňuje ich koordináciu.

- Riadenie robota je zverené inému zariadeniu - príkladom môže byť napríklad VPL. Ako riadiace zariadenie slúži vo väčšine prípadov PC. To posiela príkazy riadiacej jednotke, ktorá ovláda jednotlivé časti robota. Tento spôsob je preferovaný pri ovládaní skupiny robotov, ktoré majú byť synchronizované. Vysoký výkon PC umožňuje implementáciu zložitých stratégií, robotí potom vykonávajú presne priradené úlohy. Implementácia tohto spôsobu riadenia robota si vo väčšine prípadov vyžaduje dobrú znalosť programovacích techník. Výnimku tvorí MRDS, ktoré síce požaduje skopírovanie špeciálneho riadiaceho programu na NXT Brick, ale samotný ovládací program je možné vytvoriť v prostredí VPL.

3.2 Konštrukcia robota

Konštrukciu robota použitého pri testovaní aplikácie znázorňuje Obrázok 7. Robot bol vyvinutý ako súčasť diplomovej práce Jána Pastrňáka. V Tabuľke 2 sú uvedené jednotlivé parametre robota.

Nasleduje popis častí vyznačených na Obrázku 7:

A. Kolesá umiestnené na oboch stranách robota sú poháňané samostatnými motormi. Smer a rýchlosť robota je tak určená rozdielom v rýchlosti otáčania jednotlivých motorov.

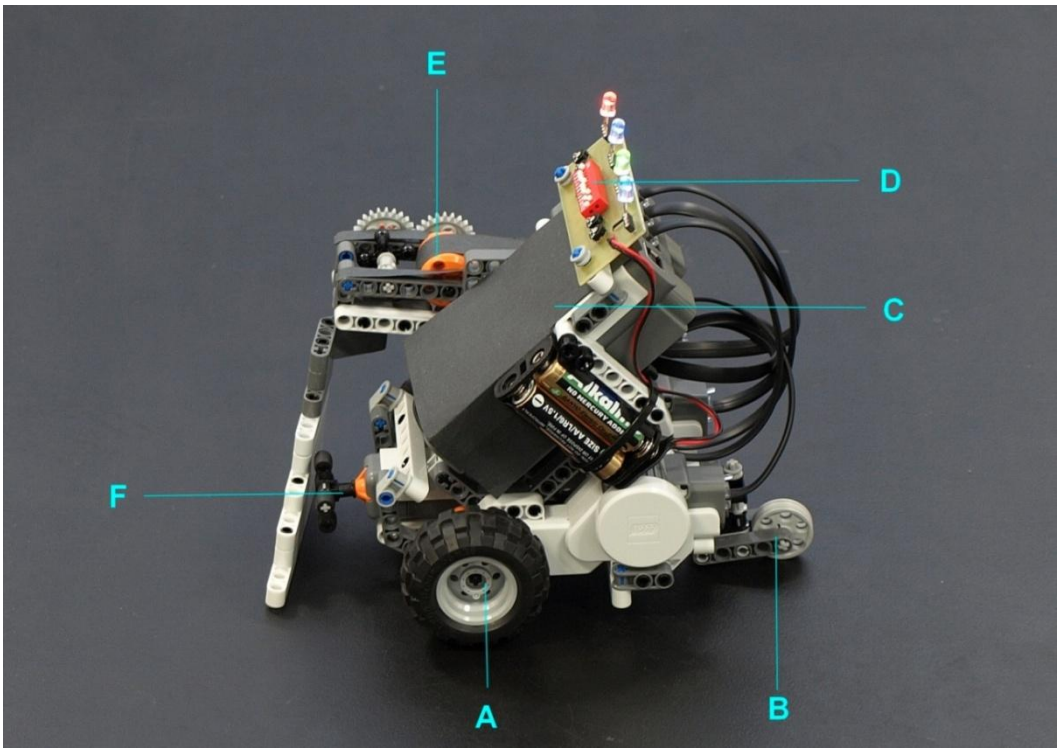
B. Malé koleso umiestnené v zadnej časti robota poskytuje väčšiu stabilitu. Toto koleso nie je poháňané motorom a otáča sa v smere pohybu robota.

C. Riadiaca jednotka NXT Brick je umiestnená v strede robota. Na tomto obrázku je zakrytá čiernou fóliou

D. Modul slúžiaci na identifikáciu robota nie je súčasťou stavebnice. Bol vyvinutý samostatne a je bližšie popísaný v kapitole 5.

E. Rameno robota, ktoré je vybavené motorom a je umiestnené na pravej strane robota. Pomocou prevodov sa na rameno upevňuje hokejka. Toto riešenie umožňuje robotovi odpaľovať loptičku.

F. V prednej časti robota je umiestnený dotykový senzor. Po odpálení loptičky tento senzor kontroluje, či sa hokejka vrátila do správnej polohy.



Obrázok 7: Robot použitý pri testovaní aplikácie

Výška	160mm
Šírka	150mm
Dĺžka	200mm
Hmotnosť	800g
Rozchod kolies	87mm
Priemer kolies	56mm

Tabuľka 2: Parametre robota prevzaté z práce Jána Pastrňáka

4. Integrácia LEGO Mindstorms a MRDS

Cieľom kapitoly je priblíženie vývoja DSS služieb nutných k ovládaniu robotov založených na stavebnici LEGO Mindstorms NXT. V nasledujúcom texte boli použité informácie čerpané z MSDN stránky Robotics Studia [1].

4.1 Pripravené služby pre ovládanie LEGO Mindstorms

Súčasťou inštalácie MRDS je aj balíka služieb umožňujúcich ovládanie niektorých bežne dostupných robotických zariadení. Jednou z takto podporovaných platforiem je aj LEGO Mindstorms.

Balík obsahuje služby pre ovládanie všetkých základných komponent (NXT Brick, senzory, motory) stavebnice. Okrem nich je pripravených aj niekoľko služieb pre ovládanie senzorov dodávaných tretími stranami. Všetky pripravené služby majú svoju reprezentáciu aj v prostredí VPL, vďaka čomu je možné vytvárať programy ovládajúce robotov založených na Mindstorms NXT aj v tomto prostredí.

Dodávané služby vyžadujú jednoduchú konfiguráciu, čím sa stávajú prístupné začiatočníkom. Na druhú stranu však poskytujú dostatok funkcionality, aby s ich použitím bolo možné vytvárať náročnejšie aplikácie (alebo služby). Okrem toho poskytujú ukážku, akým spôsobom implementovať služby pre danú robotickú platformu.

Balík služieb je rozširovaný s pribúdajúcimi zariadeniami, v dobe písania tohto textu obsahoval pätnásť služieb. Pri vytváraní služby na ovládanie robota z nich bolo niekoľko použitých.

4.2 Služba LEGO NXT Brick (v2)

Prepojenie PC a NXT brick je možné iba s využitím Bluetooth. Okrem toho nie je možné spustiť .Net framework na ovládacej jednotke stavebnice. Ako riešenie ponúka Microsoft službu, ktorá zabezpečuje komunikáciu medzi PC a NXT Brick.

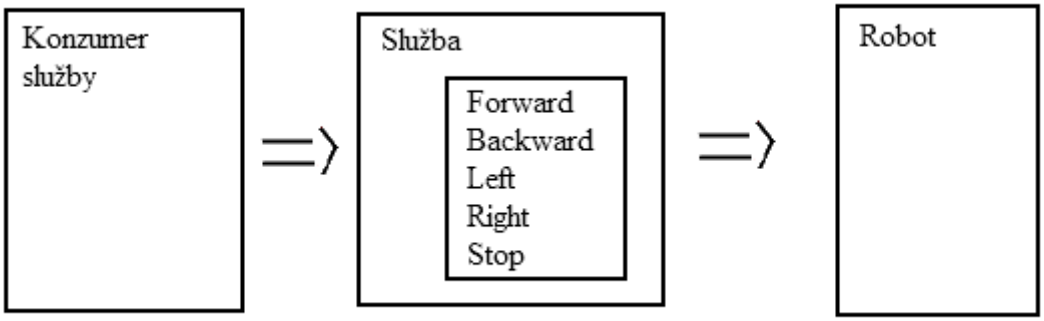
Komunikáciu zo strany NXT Brick zabezpečuje program, ktorý sa do pamäte jednotky nahrá pri prvom vytvorení spojenia s PC. Pre jeho správne fungovanie sa odporúča update firmware riadiacej jednotky na poslednú dostupnú verziu.

Komunikáciu zo strany PC má na starosti služba LEGO NXT Brick (v2), ktorá má na starosti ovlácanie riadiacej jednotky. Služba umožňuje odosielanie príkazov a prijímanie správ pomocou Bluetooth. Ostatné služby, ktoré ovládajú LEGO komponenty, majú túto službu vždy uvedenú ako partnera. Všetká komunikácia s riadiacou jednotkou sa tak uskutočňuje vždy iba pomocou tejto služby.

4.3 Ovládanie robota

Súčasťou práce bolo aj vyvinutie služby, ktorá umožňuje ovládanie robota. Vyvinutá služba poskytuje rozhranie, na ktoré sa môžu pripojiť iné služby. Ide teda o akýsi ovládací adaptér robota.

Po vytvorení ovládacej služby je automaticky zobrazená ovládacia obrazovka, ktorá simuluje konzumenta služby. Stlačenie ovládacích prvkov odosiela príkaz službe. Takéto riešenie simuluje skutočnú interakciu medzi službami. V prípade, že bude v budúcnosti vyvinutá služba prepájajúca aplikáciu identifikujúcu robotov a túto ovládaciu službu, je vhodné odstrániť ovládaciu obrazovku.



Obrázok 8: Interakcia medzi službou, jej konzumentom a robotom

4.4 Implementácia služby pre ovládanie robota

Nasledujúca kapitola popisuje kroky, ktoré je nutné urobiť pre implementáciu DSS služieb ovládajúcich robota založeného na LEGO NXT.

Pri vytváraní služieb bolo využité prostredie Visual Studio 2008 Express, v iných verziách prostredia sa názvy využitých šablón môžu mierne odlišovať. Princíp vytvárania DSS služby však ostáva rovnaký nezávisle na prostredí.

Pri vytváraní služieb je vhodné najprv zadefinovať požadované správanie služby. Obrázok 8 znázorňuje požadované interakcie medzi službou robotom a prípadnými konzumentami služby. Tento krok umožňuje špecifikáciu obslužných metód, ktoré je nutné implementovať.

Prvým krokom samotnej implementácie je vytvorenie nového projektu. V prvej verzií MRDS bolo nutné použiť príkazový riadok, vďaka integrácií s prostredím Visual Studio (ďalej iba VS) stačí vybrať šablónu určenú pre vytvorenie DSS služby. V použitom prostredí išlo o šablónu „DSS Service 2.1“.

Nový projekt obsahuje základný manifest služby a niekoľko súborov, ktoré obsahujú predgenerovaný kód vo zvolenom programovacom jazyku slúžiaci, ako základ pre implementáciu.

Ďalším krokom je špecifikácia partnerských služieb. Docieli sa importovaním knižníc obsahujúcich popis týchto služieb. Pre túto konkrétnu implementáciu boli importované služby Drive, Motor a ContactArray. Tie popisujú jednotlivé hardwarové komponenty robota. Pri importovaní knižníc je vhodné importovať tzv. generické služby. Tie majú definované

rozhranie, ale v knižnica neobsahuje ich konkrétnu implementáciu. Na generické služby je potom v manifeste možné naviazať konkrétne služby. Ide tak o akúsi obdobu Interface známu napríklad z jazyka C#. Takéto riešenie umožní vytvorenie služby, ktorá môže ovládať viacero typov robotov. Niektoré služby nemajú generickú implementáciu, ale v prípade, že generická služba existuje, je vhodné ju využiť.

```
using drive = Microsoft.Robotics.Services.Drive.Proxy;
using motor = Microsoft.Robotics.Services.Motor.Proxy;
using contactArray = Microsoft.Robotics.Services.ContactSensor.Proxy;
```

Výpis 2: Ukážka importovaných knižníc

Po naimportovaní knižníc je nutné vytvoriť porty, ktoré sú namapované na Main Porty partnerských služieb. Pomocou nich dokáže služba odosielať prožiadavky svojim partnerom.

```
[Partner("Drive",
   Contract = drive.Contract.Identifier,
   CreationPolicy = PartnerCreationPolicy.UseExisting)]
private drive.DriveOperations _drivePort = new drive.DriveOperations();
```

Výpis 3: Vytvorenie portu so službou Drive

Porty sú jednosmerné, prípady, keď je nutné správy aj prijímať, si vyžadujú pripojenie na Notifikačný port partnera. To sa docieli vytvorením špeciálneho portu určeného na prijímanie správ. Po tom je nutné vygenerovať správu Subscribe, ktorá oznámi partnerskej službe, že má odosielať správy o zmene svojho stavu na žiadaný port.

Po vytvorení spojení s partnerskými službami je ďalším krokom špecifikácia rozhrania služby. Tá pozostáva vo vytvorení objektov označujúcich jednotlivé typy správ. Po ich vytvorení je ešte nutné upraviť PortSet hlavného portu služby, aby obsahoval nové typy správ.

Posledným krokom je implementácia obslužných metód pre všetky prijímané typy správ. Oblužné metódy sú public metódy služby, ktoré je potrebné označiť atribútom ServiceHandler. Ako parameter obslužnej metódy sa uvádza typ správy, ktorý obsluhuje. V prípad, že služba prijíma napríklad správu Stop, je nutné vytvoriť obslužnú metódu, ktorá bude mať parameter Stop.

Po implementácií služby, ktorá sa odkazuje na generické služby, je nutná úprava jej manifestu a namapovanie generických služieb na konkrétne. Tento krok je možné urobiť aj v prostredí VS s využitím XML editoru. Výhodnejšie je však manifest otvoriť nástrojom Manifest Editor. Ten dokáže z implementácie služby (musí byť skompilovaná) zistiť, akých generických partnerov služba využíva. Potom stačí vybrať vhodné konkrétne služby zo zoznamu služieb ponúkaných v menu.

```
[ServicePort]
public class RobotControlServiceOperations : PortSet<DsspDefaultLookup,
        DsspDefaultDrop,  Get,  Subscribe,  Forward,
        Backward, Left, Right,          Stop>   {   }

.....

public class Forward : Submit<ForwardRequest,
PortSet<DefaultSubmitResponseType, Fault>>
{
    public Forward()
      : base(new ForwardRequest())
    {
    }
}

 [DataContract]
public class ForwardRequest { }
```

Výpis 4: Špecifikácia rozhrania služby a ukážka implementácie správy Forward

4.5 Možnosti pre ďalšie rozšírenia

Vhodným rozšírením stávajúcej funkcionality je vytvorenie služby prepájajúcej službu pre ovládanie robota a aplikácie lokalizujúcej polohu robotov. Takáto služba by mala byť schopná ovládať určitý počet robotov, pričom by mala neustále sledovať ich polohu. Pre ovládanie robotov by používala vytvorenú službu, ktorú by mala uvedenú, ako partnera.

5 Identifikácia robota na ihrisku

Táto časť diplomovej práce je venovaná problematike označovania robotov a ich následnému vyhľadávaniu. Poznatky uvedené v tejto kapitole boli čerpané z diplomovej práce Jána Pastrňáka [3].

5.1 Poloha a identita robota na ihrisku

Algoritmy, ktoré riešia stratégie hry robotov, vyžadujú znalosť polohy jednotlivých hráčov na ihrisku. Pri implementácii riešenia, ktoré má lokalizovať polohu robotov, treba zobrať do úvahy, že samotná znalosť polohy robota nestačí. Jednotlivých hráčov je potrebné nejako rozlíšiť (priradiť im unikátne ID). Bez možnosti presne určiť, ktorý robot sa nachádza na akej pozícii, by algoritmus stratégie nebol schopný správne rozdeliť príkazy.

Unikátne označenie by malo obsahovať informáciu o tom, do akého tímu daný robot patrí. Okrem toho by roboti mali byť, v rámci tímu, jednoznačne identifikovateľní. Analógiou môže byť napríklad hokej, kde dres hráča predstavuje identifikátor tímu a číslo hráča je jeho unikátny identifikátor v rámci tímu. Dres a číslo spolu tvoria unikátne ID hráča na ihrisku.

Jednou z možností, ako identifikovať robotov je zapísanie ID priamo do ovládacieho software robota. Riadiaca aplikácia potom dokáže určiť s ktorým robotom komunikuje. Na získanie polohy je možné využiť napríklad GPS. V prípade malého ihriska je však tento spôsob značne nepresný vzhľadom na obmedzenia technológie GPS. Inou možnosťou je výpočet polohy robota z jeho štartovacej pozície a prejdenej dráhy. Výpočet nemusí vykonávať priamo robot, môže sa oň postarať aj riadiaca aplikácia. Obmedzením tohto prístupu je istá nepresnosť použitých senzorov.

Druhou možnosťou je vonkajšie označenie robota a vytvorenie systému, ktorý dokáže na základe takéhoto označenia robota lokalizovať. Takéto riešenie bolo použité aj v prípade tejto diplomovej práce. Aplikácia využíva kameru, ktorá sníma celé ihrisko. Na základe prijatého obrazu potom určuje poloha a identita jednotlivých robotov.

5.2 Použité riešenie pre určenie identity a polohy robota

Toto riešenie bolo pôvodne navrhnuté Jánom Pastrňákom. Spočíva v rozlišovaní robotov na základe farebného kódu. Program potom v obraze získanom z kamery identifikuje dané kódy a na ich základe určuje pozíciu a identitu robota.

Farebný kód je tvorený štyrmi pozíciami, pričom na každú pozíciu je možné umiestniť inú farbu. Pri tvorbe kódu bola zavedená podmienka, že farba, ktorá sa nachádza na prvej pozícii kódu, bude referenčná. Táto farba sa už v kóde nemôže vyskytovať na inej pozícii. Vďaka tejto podmienke je možné prečítať farebný kód aj v prípade, že je robot ku kamera natočený v rôznych uhloch. Zároveň táto podmienka značne obmedzuje počet výsledných kódov, ktoré je robotom možné priradiť. Toto však nie je príliš veľká nevýhoda, pretože kódov je stále dostatok. Poprípade stačí zvýšiť počet miest v kóde na päť, čím sa počet možných kombinácií značne zvýši.

Posledná pozícia označuje tím, do ktorého hráč patrí. Pri hre dvoch tímov stačia dve farby. Spolu s referenčnou farbou sú teda v kóde použité tri farby.

Druhá a tretia pozícia slúžia na identifikáciu hráča v rámci tímu. Pri použití dvoch farieb dostávame štyri možné kombinácie (2^2). Na ihrisku teda v jednom okamihu môže byť osem robotov rozdelených do dvoch tímov.

Pre určenie čísla robota v rámci tímu bola modrej farbe priradená logická nula a zelenej farbe logická jednotka. Číslo robota je potom potom dostaneme prevodom druhej a tretej pozície z dvojkovej sústavy.

Obrázok 9 predstavuje diagram zobrazujúci možné kombinácie farieb v kóde a význam jednotlivých pozícií.

Obrázok 9: Diagram zobrazujúci použitie farieb

5.2.1 Hardwarové riešenie identifikácie

Za označenie robota je zodpovedný modul tvorený lisovaným spojom a štyrmi farebnými LED diódami. Bližšiu hardwarovú špecifikáciu je možné nájsť v diplomovej práci Jána Pastrňáka. Modul nebol pre potreby tejto diplomovej práce nijako upravovaný.

5.2.2 Poznámky k riešeniu

Systém identifikácie robotov pomocou LED diód so sebou prináša niekoľko problémov, tie treba pri vývoji aplikácie, ktorá analyzuje obraz získaný z kamery, brať do úvahy.

Prvým problémom je zvláštne správanie sa LED diód. Pri tomto jave stred diódy svieti bielym svetlom, zvolená farba je viditeľná až na okrajoch. Najviac je týmto javom postihnutá modrá farba. V niektorých prípadoch sa môže stať, že modrá farba nebude pre algoritmus vôbec viditeľná.

Ďalší problem predstavuje relatívne malá veľkosť diód oproti celkovej veľkosti obrazu zachyteného kamerou. Diódy sa tak môžu pri nízkom rozlíšení kamery splývať (toto sa deje v prípade, že kamera zachytáva ihrisko v príliš ostrom uhle).

Posledným problémom, na ktorý som narazil je, že pri vysokom jase obrazu sa niektoré časti robota môžu pre algoritmus javiť ako diódy. Tento jav sa najviac prejavuje u červenej farby, čo je pochopiteľné, pretože červená je relatívne často používana farba pre súčiastky LEGO Technics.

Popísané problémy sa algoritmus vyhľadávania snaží riešiť kontrolou veľkosti a tvaru diód. Okrem toho prebieha sekundárna filtrácia nájdených bodov, kde sú vylúčené body, ktoré sú príliš vzdialené od ostatných.

6 Vplyv kamery a výšky robota na riešenie vyhľadávania robotov

Pre správne určenie polohy robota z obrazu získaného kamerou je nutné pochopiť, akým spôsobom ovplyvňuje pozícia a parametre kamery získaný obraz. Nasledujúca kapitola približuje, akým spôsobom kamera obraz zachytáva a čo z toho plynie pre algoritmus vyhľadávania robotov. S vplyvom kamery úzko súvisí aj to, ako závisí určovanie pozície robota od jeho výšky. Popisu tohto problému je venovaná predposledná podkapitola tejto časti. Poznatky uvedené v tejto kapitole boli čerpané zo skrípt k predmetu Digitální spracování obrazu [4], ako aj z dokumentácie OpenCV [5]. Niektoré matematické konštrukie boli čerpané zo stránok wikipédie venujúcich sa danej problematike [6].

6.1 Homogénny súradnicový systém

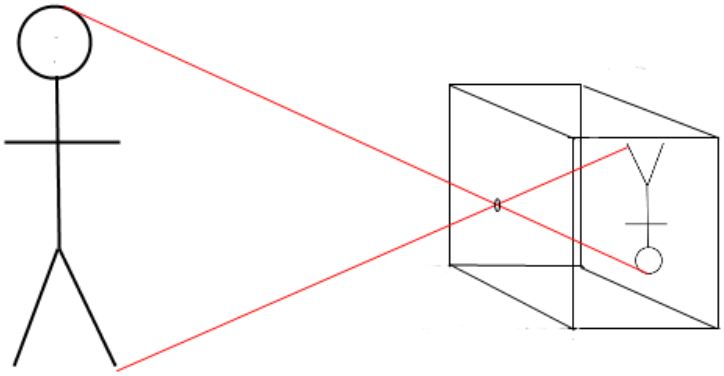
V nasledujúcich kapitolách sa bude často vyskytovať pojem homogénny súradnicový systém (poprípade homogénne súradnice). Tento súradnicový systém sa v projektívnej geometrii využíva namiesto klasického karteziánskeho systému. Jeho hlavnou výhodou je, že umožňuje zobrazovať aj body, ktoré ležia v nekonečne pomocou konečných súradníc. Pre projektívnu geometriu (ktorá je využitá pre rekonštrukciu 3D obrazu), to predstavuje kľúčovú výhodu, pretože sú tým umožnené operácie s maticami, ktoré by neboli možné pri použití bodov v nekonečne.

Homogénne súradnice popisujú bod v n-rozmernom afinnom priestore ako smer v pridruženom (n+1)-rozmernom vektorovom priestore. Ako príklad môžeme uvažovať dvojrozmerný afinný priestor. Homogénne súradnice bodu *(x,y)*, patriaceho do daného priestoru, bude trojica súradníc *(wx,wy,w)*. Ako *w* je možné zvoliť ľubovoľné reálne číslo, ktoré však musí byť rôzne od nuly. V homogénnom súradnicovom systéme je teda bod reprezentovaný nekonečným počtom vektorov patriacich pridruženému vektorovému priestoru. V praxi je vhodné voliť *w=1*, výsledná trojica je potom *(x,y,1)*. Prevod z homogénnych do afinných súradníc docielime opačným postupom, teda vydelením jednotlivých zložiek vektora poslednou zložkou.

6.2. Vplyv kamery na získaný obraz

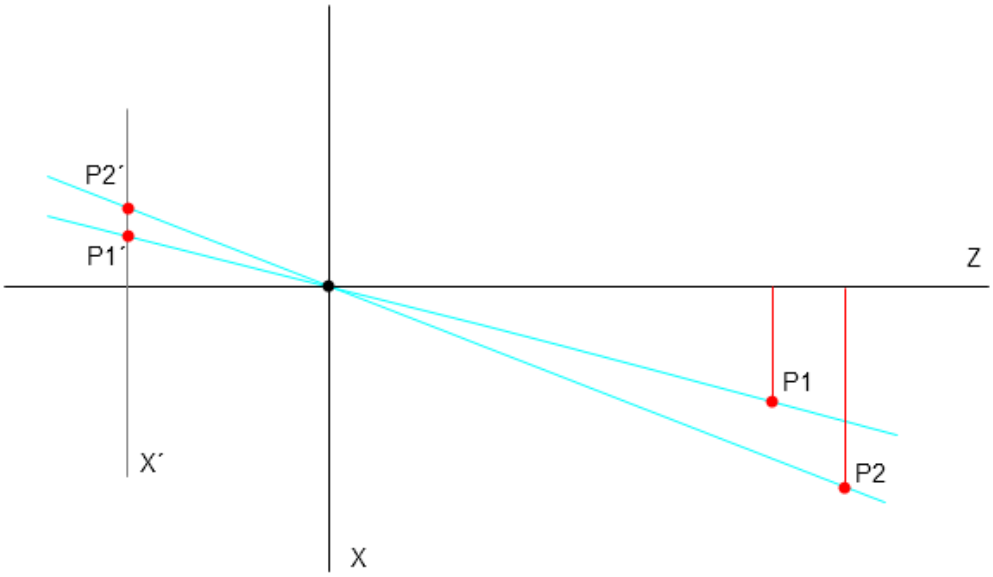
Vzťah medzi objektom v reálnom svete (3D priestor) a jeho obrazom získanom kamerou (2D priestor) popisuje tzv. pinhole camera model. Kamera sa v tomto prípade modeluje ako krabica, ktorá má v jednej stene dieru. Objekty sa potom cez túto dieru premietajú na zadnú stenu krabice. V prípade základného modelu sa uvažuje o ideálnej kamere. Pre túto kameru predstavuje dieru práve jeden bod, cez ktorý prechádzajú všetky lúče svetla.

V skutočnom svete je model kamery komplikovanejší. Obraz je okrem skreslenia spôsobeného pinhole modelom skreslený aj vplyvom šošoviek objektívu. Tieto skreslenia sú zrejmé hlavne na okrajoch obrázku. Napriek tomu poskytuje daný model dobrú predstavu o fungovaní kamery.



Obrázok 10: Pinhole camera model

Z Obrázku 11 je zrejmé, že obraz zachytený kamerou neposkytuje úplne presné informácie o skutočnom objekte. Zobrazenie z 3D do 2D, ktoré predstavuje pinhole camera model je možné popísať ako perspektívnu projekciu, ktorú nasleduje otočenie o 180˚. Samotné otočenie nie je príliš dôležité, pretože býva riešené kamerou (výstup kamery je automaticky otočený). Pre analýzu polohy robota je dôležité si uvedomiť, aký vplyv má perspektívna projekcia na získaný obraz.



Obrázok 11: Geometrické zobrazenie pinhole camera modelu

Perspektívna projekcia je jednou z 3D projekcií, teda metód pre mapovanie 3D bodov do 2D roviny (image plane, zobrazovacia rovina). Body sa na zobrazovaciu rovinu premietajú podľa priamok prechádzajúcich jedným bodom, nazývaným centrum projekcie. V prípade kamery je centrom projekcie jej objektív. Dôsledkom tohto spôsobu premietania je, že objekty vzdialenejšie od centra premietania sa premietnu na menšiu plochu, ako objekty k centru bližšie.

Tento jav je dobre viditeľný na Obrázku 11, kde vidno, že pomer vzdialeností bodov P1 a P2 od osi Z nezostal v premietnutom obraze zachovaný.

Pre vyhľadávanie polohy robotov vzhľadom na ihrisko je dôležité eliminovať vplyv, ktorý má perspektívna projekcia na získaný obraz. V aplikácii je tento problém riešený transformáciou, ktorá prevádza obraz do požadovaného (nedeformovaného transformáciou) tvaru.

6.2.1 Matematický model kamery

Matematicky sa zobrazenie, ktoré vykonáva kamera, dá popísať nasledujúcim vzťahom:

$$sm' = A[R|t]M$$

Pre lepšie pochopenie je dobré rozpísať jednotlivé premenné, dostaneme nasledujúcu rovnicu:

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- M predstavuje bod v 3D priestore, zapísaný pomocou homogénnych súradníc.

- Matica A sa nazýva maticou vlastných parametrov (matrix of intrinsic parameters). Popisuje, aký vplyv majú paramatre kamery na obraz. (c_x,c_y) predstavujú centrum projekcie, vo väčšine prípadov ide o stred obrazu. f_x a f_y sú ohniskové vzdialenosti kamery.

- Matica [R|t] sa nazýva tiež maticou vonkajších parametrov (extrinsic). Táto matica popisuje transformáciu obrazu, ktorá sa vykonáva vzhľadom na pozíciu kamery.

- m´ sú súradnice bodu premietnutého do 2D priestoru. Tieto súradnice sú taktiež zapísané v homogénnom súradnicovom systéme.

Matica A sa nemení so zmenou pozície kamery (popípade so zmenou snímanej scény), mení sa len v prípade, keď priamo modifikujeme parametry kamery (napríklad zaostrovanie). Matica vonkajších parametrov sa mení podľa pozície kamery vzhľadom na snímanú scénu.

Vynásobením matíc A a [R|t] získame maticu kamery (camera matrix). Využitie tejto matice je vhodné v prípadoch, že sa nemení nastavenie kamery a kamera sa nepohybuje okolo snímanej scény.

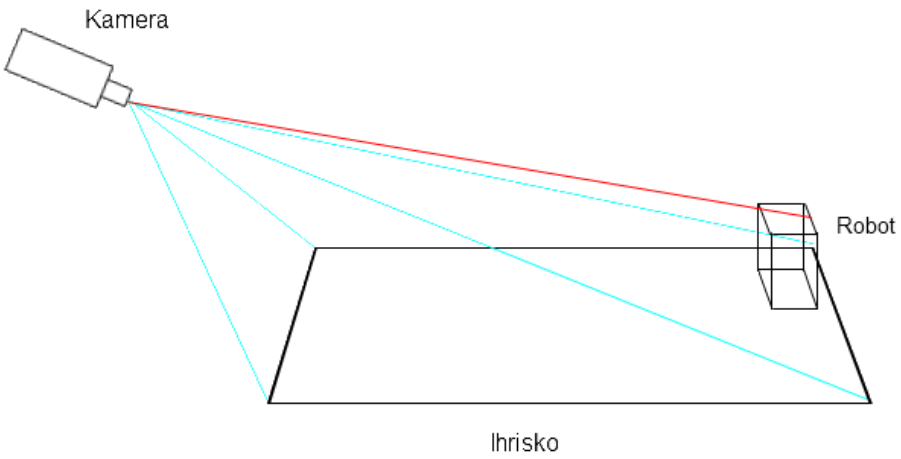
6.2.2 Využitie v aplikácii

Pri vyhľadávaní pozície robotov sa využíva znalosť popísaných vzťahov hlavne v kalibračnej časti. Keďže kamera snímajúca ihrisko je stacionárna a v priebehu snímania obrazu sa nezaostruje, je výhodné získať maticu kamery raz a ďalej s ňou už iba pracovať. V prípade, že by sa kamera okolo ihriska pohybovala (poprípade by sa menilo zaostrenie), bolo by nutné vyhodnocovať jednotlivé parametre vždy pred výpočtom pozície robotov. Takéto riešenie by však bolo značne náročné na výpočetný výkon.

6.3 Vplyv výšky robota na výpočet jeho pozície

Keďže pozícia robota sa vypočítava z pozície LED diód nájdených v obraze, je nutné brať do úvahy umiestnenie diód na robotovi. Modul obsahujúci diódy je umiestnený na vrchu robota (viď Obrázok 4). Kamera snímajúca ihrisko môže byť umiestnená na jednej strane ihriska. Z definície pinhole modelu kamery je zrejmé, že výška robota bude mať vplyv na zobrazenie LED diód v zachytenom obraze.

Na Obrázku 11 je znázorňuje spôsob, akým vplýva výška robota spolu s umiestnením kamery na konečnú pozíciu diód v obraze. Na obraze zachytenom kamerou sa bude pozícia diód javiť mimo hraciu plochu, aj keď je robot v skutočnosti umiestnený na jej okraji. Ak by bol tento jav neošetrený by algoritmus nebol schopný nájsť presnú polohu robota. Presnosť nájdenej pozície by tak záležala na umiestnení kamery. V prípade, že by kamera snímala obraz z hora, by boli zistené pozície relatívne zodpovedali skutočnému umiestneniu robota, aj keď presnosť by klesala so vzdialenosťou robota od kamery. Pri umiestnení kamery podobne, ako na Obrázku 12 by boli všetky pozície nesprávne určené.



Obrázok 12: Interakcia medzi pozíciou kamery a výškou robota

6.4 Zhrnutie poznatkov

Z poznatkov uvedených v tejto kapitole vyplýva, že pri identifikácii robota a určení jeho pozície nepostačuje nájsť pozíciu LED diód v obraze. Keďže sú diódy umiestnené na vrchu robota je nutné zobrať do úvahy aj výšku robota. Okrem toho treba vyriešiť deformácie spôsobené projektívnou transformáciou.

7 Aplikácia pre určovanie pozície a ID robota v obraze

Z predchádzajúcich kapitol je zrejmé, že algoritmus určujúci pozíciu robota vzhľadom na ihrisko musí riešiť niekoľko problémov:

- Vyhľadanie LED diód v obraze.
- Elimináciu skreslenia vzdialeností vplyvom perspektívnej projekcie.
- Elimináciu posunutia spôsobeného umiestnením LED diód na vrchu robota.
- Získanie správneho otočenia robota a jeho ID.

Nasledujúca kapitola bude venovaná praktickému riešeniu týchto problémov. Niektoré informácie uvedené v kapitole boli čerpané zo skrípt pre predmet Digitální spracování obrazu [4] a Počítačová Grafika II [7]. Zdrojom informácií týkajúcich sa knižnice Emgu CV bola stránka venovaná tomuto projektu [8].

7.1 Predstavenie knižnice Emgu CV

Pri vytváraní aplikácie bola využitá knižnica Emgu CV. Ide o voľne šíriteľný .Net wrapper knižnice OpenCV, určený pre uľahčenie práce s grafikou. Emgu CV umožňuje volania funkcií implementovaných v OpenCV z jazykov kompatibilných s prostredím .Net (napríklad C#, IronPython alebo VB).

Táto knižnica nie je jediná svojho druhu, oproti podobným projektom má však niekoľko výhod, ktoré boli rozhodujúce pri jej výbere:

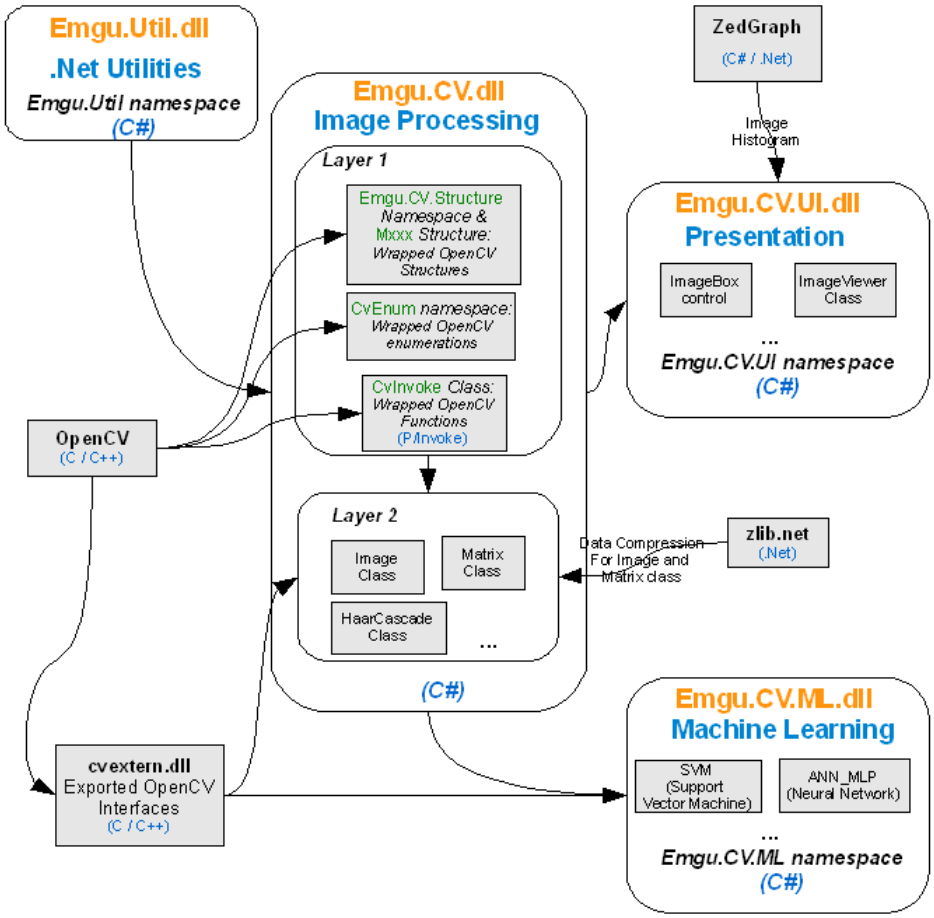
- Aktívna komunita - vývoju sa venuje početná a aktívna komunita vývojárov. Stránka projektu obsahuje veľa návodov ako aj kompletnú dokumentáciu API. Návody sú dostupné vo viacerých podporovaných jazykoch.
- Knižnica je celá napísaná v jazyku C# - vďaka tomu, že Emgu CV nevyužíva potencionálne nebezpečný kód (unsafe code, patria tu napríklad priame volania pointrov alebo priame odkazovanie na C++ knižnicu) je možné túto knižnicu kompilovať aj v prostredí Mono

7.1.1 Architektúra Emgu CV

Knižnica sa dá rozdeliť do dvoch vrstiev. Prvá vrstva obsahuje štruktúry a funkcie, ktoré priamo zodpovedajú funkciám implementovaným v OpenCV. Nad touto základnou vrstvou je postavená rozširujúca vrstva, ktorá umožňuje využitie výhod poskytovaných prostredím .Net. Príkladom môže byť napríklad generická trieda Image<TColor,TDepth>

umožňujúca jednoduché definovanie obrázku s využitím jedného z farebných modelov (RGB, HSV) a prevody medzi týmito modelmi.

Obrázok 13, prevzatý zo stránky Emgu CV [5], zobrazuje bližší pohľad na architektúru knižnice.



Obrázok 13: Architektúra Emgu CV, ako je uvedená na stránke projektu [5]

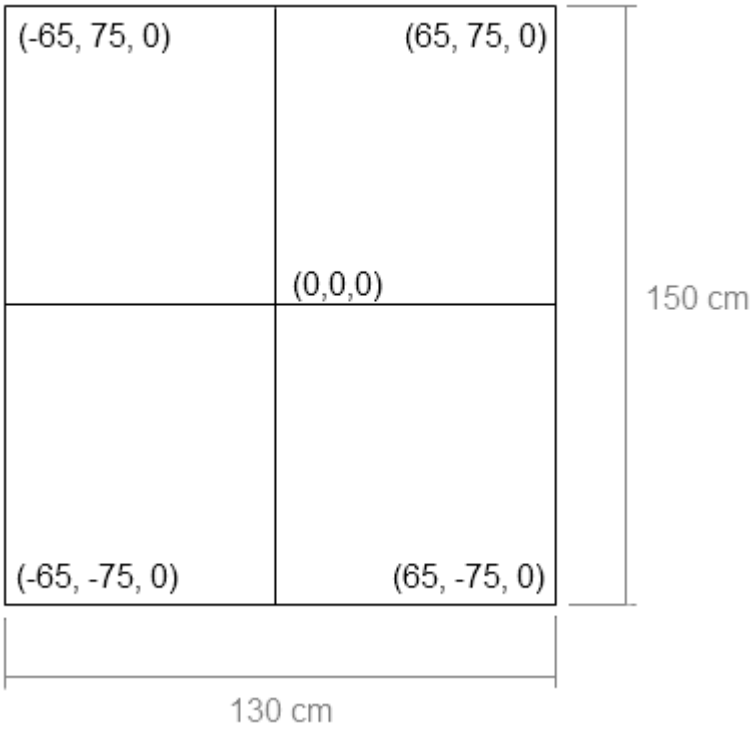
7.1.2 Prečo používať wrapper

V prípade priameho použitia knižnice OpenCV by musela byť aplikácia vytvorená v nespravovanom C++ (unmanaged C++). Prostredie .Net síce takéto riešenie podporuje, ale mohlo by neskôr spôsobiť problémy pri pokuse o prepojenie s časťou aplikácie zodpovednou za riadenie robotov. Využitie kódu natívneho pre .Net ďalej umožňuje využitie možností, ktoré toto prostredie prináša. Príkladom sú už spomínané generické implementácie viacerých štruktúr a jednoduchšia správa pamäte.

Nevýhodu je, že kód napísaný s použitím Emgu CV nebude nikdy tak výkonný, ako kód využívajúci priamo OpenCV. Preto bolo jednou z priorít pri tvorbe aplikácie čo najviac optimalizovať algoritmy.

7.2 Voľba súradnicového systému

Voľbou súradnicového systému sa rozumie, ako budú reprezentované body v reálnom svete. Pre správne fungovanie programu je dôležité, aby používané označovanie bodov reálneho sveta bolo jednotné. Preto bola zavedená konvencia, že stred ihriska má súradnice (0,0,0). Tým pádom sa stred súradnicového systému nachádza v strede ihriska. Šírka ihriska je totožná s osou X, výška je potom totožná s osou Y. Takéto označenie umožňuje jednoduché vypočítanie polohy rohov ihriska, keďže rozmery ihriska sú známe.



Obrázok 14: Rozmery ihriska spolu so súradnicami rohov ihriska

Na obrázku 14 sú vyznačené rozmery ihriska, spolu so súradnicami rohov ihriska. V aplikácii sú rozmery ihriska zadané v konfigurácii, pričom súradnice rohov sa z nich vypočítavajú. Pri zmene ihriska tak stačí upraviť jeho rozmery v konfigurácií.

7.3 Vyhľadávanie farebných škvŕn v obraze

Vyhľadanie farebných škvŕn v obraze je prvým krokom, ktorý treba urobiť, pre lokalizáciu jednotlivých robotov. Pri vytváraní aplikácie vzniklo niekoľko verzií algoritmu,

ktorý rieši tento problém. Jednotlivé verzie sa odlišovali napríklad voľbou farebného modelu (RGB, HSV) a možnosťani nastavenia parametrov.

Nasledujúca časť sa venuje popisu algoritmu, ktorý bol využitý vo finálnej verzii.

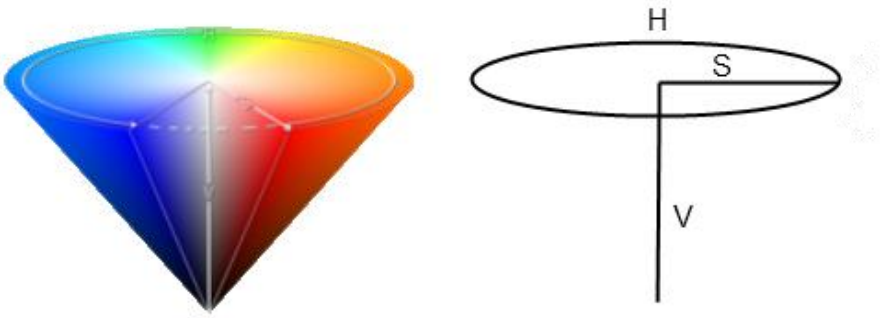
7.3.1 Voľba farebného modelu

Finálna verzia využíva farebný model nazývaný HSV (Hue Saturation Value). Je to model, kde je farba reprezentovaná troma zložkami.

- Hue - farebný tón, ktorého hodnota môže byť od 0˚ až do 360˚. Udáva prevládajúcu farbu. Reprezentácia farebného tónu je kruhová, čo znamená, že napríklad odtiene červenej spadajú do intervalov približne od <0˚,30˚> a <330˚,360˚>. Túto skutočnosť treba brať do úvahy pri vytváraní vhodných parametrov pre vyhľadávanie farieb.

- Saturation - táto hodnota predstavuje sýtosť farby. Sýtosť farby sa vyjadruje pomerom šedej farby k farebnému tónu. Meria sa v percentách, kde 0% predstavuje šedú farbu a 100% úplne sýtu farbu.

- Value - jas. Táto hodnota predstavuje relatívnu svetlosť, alebo tmavosť farby. Podobne, ako sýtosť, sa udáva v percentách.



Obrázok 15: Kužeľová reprezentácia modelu HSV

Výhodou tohto modelu oproti modelu RGB je, že umožňuje lepšiu špecifikáciu farieb, ktoré sa majú v obraze vybrať. Toto je dôležité v prípade nepriazdnivých svetelných podmienok.

7.3.2 Algoritmus pre vyhľadávanie škvŕn

Algoritmus, ktorý vyhodnocuje farebné škvrny (blob) v obraze sa skladá z nasledujúcich krokov:

1. Získanie obrazu, ktorý sa má vyhodnotiť.
2. Získanie parametrov vyhľadávania.
3. Rozdelenie na farebné hladiny zodpovedajúce modelu HSV.
4. Vytvorenie masky pre Saturation a Value.
5. Výber potencionálnych škvŕn na základe ich farebného tónu (Hue).
6. Aplikácia masky na potencionálne škvrny.
7. Filtrácia škvŕn na základe parametrov.

V aplikácii je za lokáciu škvŕn zodpovedná trieda BlobFinder. Tá dostáva obraz, v ktorom je nutné nájsť farebné škvrny, ako parameter konštruktora (aplikácia obraz získava buď načítaním zo súboru, alebo zachytením obrazu z kamery). Parametre pre vyhľadávanie sú tvorené množinou prípustných farieb. Ako druhotný parameter slúži ohraničenie, v ktorom sa majú škvrny vyhľadávať. Toto ohraničenie zodpovedá ploche ihriska.

Parametre pre vyhľadávanie sú získavané v kalibračnej časti aplikácie, ktorá je popísaná v prílohe venovanej testovaniu aplikácie.

Získaný obraz je rozdelený na farebné hladiny, čím vzniknú tri obrazy v odtieňoch šedej. Ide o jednokanálove reprezentácie jednotlivých zložiek modelu HSV, body patriace jednotlivým hladinám tak reprezentujú hodnoty zložiek v danom obrazovom bode.

Ďalším krokom je vytvorenie masky zloženej z vrstiev Saturation a Value. Maska reprezentuje množinu bodov, ktoré majú požadované hodnoty jasu a sýtosti. Pre vytvorenie masky sa využíva metóda ThresholdBinary, ktorá priradí bodu hodnotu 1 v prípade, že je jeho hodnota vyššia ako zadaná hranica. V opačnom prípade je bodu priradená hodnota 0. Metóda ThresholdBinary sa aplikuje na vrstvy S a V. Výsledná maska potom vzniká logickým súčtom takto získaných vrstiev. Maska sa dá zobraziť ako obraz, ktorý obsahuje biele body všade tam, kde dosahujú S a V požadované hodnoty. Ostatné miesta sú čierne.

Podobne je riešená lokalizácia bodov s vhodnými hodnotami farebného tónu (Hue). Na vrstvu zodpovedajúcu tejto zložke sa aplikuje metóda cvInRangeS, ktorá funguje podobne, ako TresholdBinary. Rozdielom je, že umožnuje špecifikáciu hodnôt intervalom. Takže metóda priradí hodnotu 1 bodom, ktoré spadajú do daného intervalu a 0 bodom mimo intervalu. Pri červenej farbe môže byť interval rozdelený na dve časti (<0˚,30˚> spolu s <330˚,360˚>) treba vyhľadať body patriace do doplnkového intervalu (<30˚,330˚,>) a získanú vrstvu potom invertovať.

Výsledná mapa škvŕn, pre určitú farbu, potom vzniká logickým súčtom masky a vrstvy získanej v predchádzajúcom kroku.

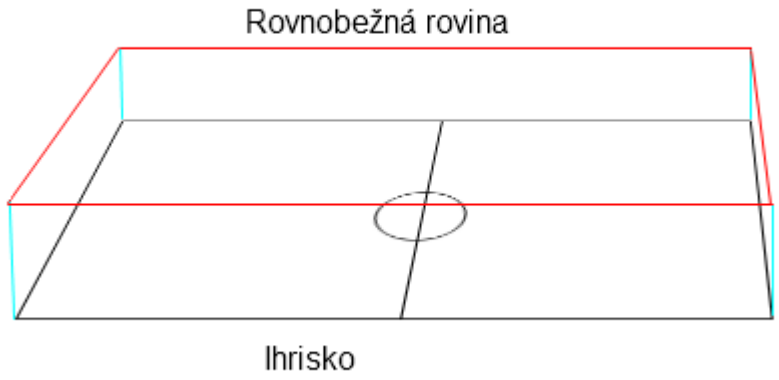
Emgu CV umožňuje rýchlú lokáciu blobov pomocou metódy FindContours. Po získaní pozícií farebných škvŕn je nutné vybrať tie, ktoré môžu predstavovať LED diódy. Kritériom sú veľkosť, pozícia a tvar. Vďaka tomu sú odstránené príliš veľké škvrny, odrazy (škvrny s príliš malou veľkosťou), body mimo ihrisko a body, ktoré nemajú požadovanú svietivosť.

Problémom riešenia je jeho citlivosť na svetelné podmienky. V šere pracuje s veľmi veľkou úspešnosťou. V prípade vysokej koncentrácie jasu v obraze môže algoritmus vyhodnotiť niektoré časti robota ako reprezentáciu LED diód. Ako už bolo uvedené, tento problém je spôsobený malou veľkosťou diód vzhľadom na ihrisko. Pri väčšej vzdialenosti kamery od ihriska (a tým aj od robota), je veľkosť diód na zachytenom obraze nedostačujúca.

7.4 Vyrovnanie sa s posunutím spôsobeným výškou robota

Cieľom nasledujúcej kapitoly je popísanie postupu, ktorý vedie k eliminácií posunutia spôsobeného umiestnením LED diód na vrchu robota. Popis tohto problému je uvedený v kapitole 6.3.

Pri riešení problému si je nutné uvedomiť, že v prípade zanedbania tohto javu, by sa predpokladalo, že LED diódy ležia v rovine, ktorú tvorí ihrisko. riešením problému je teda nájdenie roviny, ktorá je rovnobežná s ihriskom, pričom jej vzdialenosť od ihriska je rovnaká, ako výška robota. Ohraničenie tejto roviny tvoria štyri body, ktoré sú v skutočnom svete nad rohmi ihriska. Obrázok 16 ukazuje, ako môže vyzerať vzťah medzi týmito dvoma rovinami na obrázku získanom kamerou.



Obrázok 16 Rovina obsahujúca LED diódy

Algoritmus, ktorý eliminuje vplyv perspektívnej projekcie (je popísaný v nasledujúcej kapitole), vyžaduje znalosť polohy štyroch bodov v skutočnom svete a ich polohu v obrázku. Riešením posunutia je teda nájdenie štyroch bodov, ktoré zodpovedajú bodom ohraničujúcim rovinu umiestnenú nad ihriskom.

7.4.1 Možné riešenia

Jedným z navrhovaných riešení bolo umiestnenie majákov (alebo podobného dobre rozoznateľného označenia) na okraje ihriska vo výške, ktorá zodpovedá výške robota. Výhodou takéhoto riešenia je jeho relatívna jednoduchosť. Pri takomto riešení by postačovalo získať pozíciu majákov v obraze (s využitím algoritmu pre vyhľadávanie škvŕn) a pokračovať eliminációu perspektívnej projekcie. Nevýhodou riešenia je, že by vyžadovalo úpravu ihriska vždy, keď sa zmení výška robota.

Iným riešením bolo označenie bodov v správnej výške v kalibračnej časti aplikácie. Problémom takéhoto riešenia je, že správne určiť, kde sa majú dané body nachádzať, je veľmi ťažké (ide o manuálny krok, ktorý by vyžadoval správny odhad vzdialeností na obrázku).

V prípade použitého riešenia sa v kalibračnej časti aplikácia manuálne označujú rohy ihriska, rovina obsahujúca LED diódy je potom automaticky dopocítaná. Manuálne označovanie rohov je jednoduchšie, ako odhad, v akej výške sa majú nachádzať LED diódy. Okrem toho nevyžaduje úpravu ihriska.

7.3.2 Postup nájdenia roviny

Postup pre nájdenie ohraničujúcich bodov roviny je možné zhrnúť do nasledujúcich krokov:

1. Získanie súradníc rohov ihriska v skutočnom svete.
2. Získanie súradníc bodov ohraničujúcich rovinu s LED diódami v skutočnom svete.
3. Získanie súradníc rohov ihriska na obrázku.
4. Výpočet matice kamery.
5. Výpočet súradníc bodov ohraničujúcich rovinu s LED diódami na obrázku.

Prvým krokom je získanie súradníc, ktoré reprezentujú všetky potrebné body v reálnom svete. V kapitole 6.2 bol uvedený spôsob, akým sú reprezentované súradnice ihriska v 3D priestore. Poloha rohov ihriska je teda známa. Pre získanie súradníc roviny ležiacej nad ihriskom stačí pripočítať výšku robota k z-súradnici jednotlivých vrcholov.

Súradnice rohov ihriska na obrázku získava kalibračná časť aplikácie. Ide o manuálnu časť, pričom užívateľ označí rohy ihriska v obrazovke na to určenej.

Matica kamery je popísaná v kapitole 6.2. S použitím tejto matice je možné vypočítať, kde sa premietnu body ohraničujúcu vrstvu s LED diódami. Pre zistenie matíc sa využíva statická metóda CalibrateCamera, patriaca objektu CameraCalibration. Pre správne určenie dvoch matích, ktoré tvoria zložky matice kamery vyžaduje funkcia znalosť skutočnej a premietnutej polohy minimálne štyroch bodov, ktoré sú viditeľné na obrázku. Ako vstupy

slúžia 3D súradnice jednotlivých rohov ihriska a ich 2D reprezentácia získaná od užívateľa v kalibračnej časti.

Ďalším krokom je vynásobenie matice s jednotlivými bodmi ležiacimi nad rohmi ihriska. Výsledkom násobenia je reprezentácia bodov v obrázku. Body takto získané sú reprezentované v homogénnych súradniciach, preto je potrebné ich previesť do karteziánskych súradníc.

Získané súradnice bodov sú potom využité v algoritme pre vyhľadávanie škvŕn v obraze, kde slúžia ako ohraničenie oblasti, v ktorej sa majú škvrny vyhľadávať. Okrem toho sa používajú aj v algoritme, ktorý odstraňuje vplyv perspektívnej projekcie.

7.3.3 Poznámky k riešeniu

Ako už bolo uvedené v kapitole 6.2, kamera snímajúca ihrisko je statická. Z tohto dôvodu postačuje získať maticu kamery len raz. V aplikácií sa tento algoritmus volá priamo v kalibračnej časti. Vyhľadávacia časť aplikácie potom už iba využíva získanú maticu, poprípade získané body.

Riešenie by bolo možné rozšíriť o automatické vyhľadávanie rohov ihriska. V takomto prípade by bola nutná modifikácia ihriska (pridanie majákov do jeho rohov), táto modifikácia by však bola iba jednorázová (oproti umiestneniu majákov do výsky robota). Riešenie by tak poskytovalo možnosť automatickej kalibrácie kamery bez nutnosti úpravy ihriska pri zmene konštrukcie robota.

7.4. Eliminácia perspektívneho skreslenia

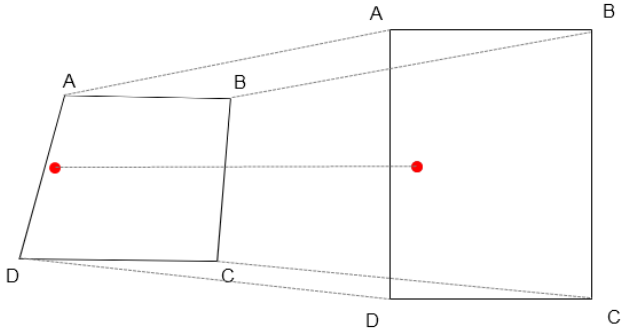
V kapitole 6.2 je popísaný vplyv, ktorý má perspektívna projekcia na získaný obraz. Eliminácia tohto javu je dôležitá kvôli presnosti, s akou dokáže aplikáciu určiť skutočnú polohu robota.

Pre elimináciu skreslenia je nutné popísať transformáciu, ktorá dokáže obraz previesť do požadovaného (neskresleného) tvaru. Pre popis transformácie sa využíva transformačná matica, ktorú je možné vytvoriť pomocou znalosti štyroch bodov v zdrojovom obraze a ich premietnutej polohy v cieľovom obraze.

Ako body v zdrojovom obraze sú použité body získané algoritmom popísaným v kapitole 6.3 (body získané kvôli eliminácií vplyvu výšky robota). Použitie týchto bodov zaisťuje, že bude transformovaná správna rovina (rovina obsahujúca LED diódy, nie rovina ihriska).

7.4.1 Projektívna transformácia

Projektívna transformácia (Projective transformation, Homography transformation) je jedna z geometrických transformácií obrazu. Zachováva linearitu objektov (čiary sa transformujú na čiary), nezachováva však uhly ani veľkosť transformovaných objektov.



Obrázok 17: Požadovaná transformácia obrazu

V počítačovej grafike často nachádza uplatnenie v prípadoch, ktoré súvisia so stredovým premietaním (prípad pinhole modelu kamery). Príkladom využitia je pridávanie 3D objektov do scény tak, aby nebolo viditeľné, že tam nepatria. Poprípade zistenie pohybu kamery z dvoch rôznych obrázkov jednej scény. Jedným z využití tejto transformácie je aj eliminácia perspektívneho skreslenia.

Vzťah popisúci projektívnu transformáciu bodu s afinnými súradnicami (x,y) sa dá vyjadriť nasledovne:

$$\begin{bmatrix} \varphi(x,y) \\ \psi(x,y) \\ \omega(x,y) \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Zo vzťahu vyplýva, že homogénne súradnice bodu po transformácií budú (φ(x,y), ψ(x,y),ω(x,y)). Pre prevod do afinných súradníc teda platí:

$$x' = \varphi(x,y)/\omega(x,y)$$

$$y' = \psi(x,y)/\omega(x,y)$$

Projektívnu transformáciu teda popisuje deväť hodnôt (p_{11}, p_{12}, ..., p_{33}). Uvedené hodnoty je možné násobiť ľubovoľným nenulovým číslom bez toho, aby sa zmenil výsledok (toto plynie z definície homogénnych súradníc, bod môže byť reprezentovaný nekonečným počtom vektorov v pridruženom (n+1)-rozmernom priestore).

K získaniu hodnôt popisujúcich transformáciu je potrebná znalosť štyroch bodov vo vstupnom obraze a ich reprezentáciu vo výstupnom obraze. Body vo vstupnom obraze môžeme označiť napríklad A'_1, A'_2, A'_3, A'_4 a body vo výstupnom obraze označíme A_1, A_2, A_3, A_4. Ďalej je nutné zaviesť označenie $p_1 = (p_{11}, p_{12}, p_{13})$, $p_2 = (p_{21}, p_{22}, p_{23})$, $p_3 = (p_{31}, p_{32}, p_{33})$ a $X_i = (x_i, y_i, 1)^T$.

Na základe vzťahu definujúceho projektívnu transformáciu tak dostávame:

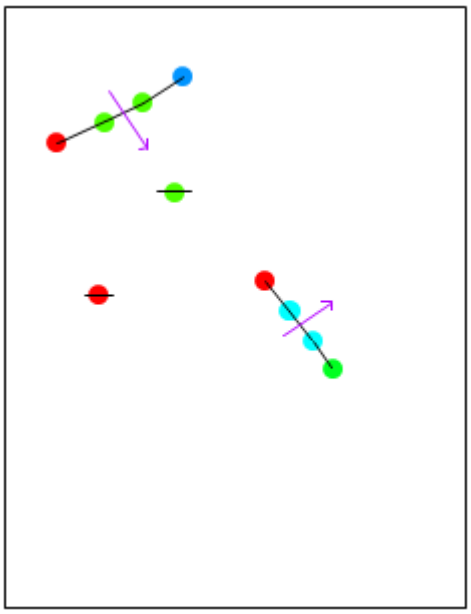
$$\begin{bmatrix} w'_i x'_i \\ w'_i y'_i \\ w'_i \end{bmatrix} = \begin{bmatrix} P_1^T \\ P_2^T \\ P_2^T \end{bmatrix} X_i$$

Z tretej rovnice vzťahu dostávame $w'_3 = p_3^T X_i$. Po dosadení do prvých dvoch rovníc a úprave získame:

$$\begin{bmatrix} X_i^T & 0 & -x'_i X_i^T \\ 0 & X_i^T & -y'_i X_i^T \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Pretože každý z bodov prispieva k nájdeniu transformačnej matice dvoma rovnicami, vzniká osem rovníc o deviatich neznámych. Vieme, že transformačnú maticu je možné násobiť ľubovolným nenulovým číslom bez zmeny výsledku, vďaka tomu je možné zvoliť, ako hodnotu jedného z prvkov predom určené číslo (napríklad 1). Úskalím tohto riešenia by bolo, ak by skutočná hodnota daného prvku bola nulová. V takom prípade by bola sústava rovníc neriešiteľná.

7.5 Určenie kódu a pozície robota a jeho otočenia



Obrázok 18: Výsledná pozícia a otočenie robotov

Po spracovaní obrazu jednotlivými algoritmami získava aplikácia množinu, ktorá obsahuje svetelné kódy bodov a ich pozíciu vzhľadom na ihrisko. Posledným krokom,

ktorý je potrebné urobiť je prečítanie kódov robotov, čím je získaná aj ich pozícia spolu s otočením.

Cieľom algoritmu je vytvorenie vhodných n-tíc (n je počet diód, ktoré boli použité k identifikácii robota, v prípade použitého označovania je n rovné štyrom) bodov, ktoré predstavujú ID robota. Po vytvorení n-tice je možné využiť znalosť pozície jednotlivých bodov predstavujúcich ID k výpočtu pozície robota a jeho otočenia.

Získaná množina môže obsahovať body, ktoré nepredstavujú diódy. Môže to byť spôsobené chybou algoritmu pre vyhľadávanie farebných škvŕn v prípade nepriazdnivých svetelných podmienok. Z tohto dôvodu je potrebná kontrola vzdialeností medzi bodmi, keďže LED diódy nemôžu byť od seba priveľmi vzdialené (ale ani príliš blízko). Algoritmus využíva aj kontrolu uhlov, ktoré zvierajú priamky prechádzajúce dvojicami bodov s osou X. V prípade, že body patria jednému robotovi by uhly nemali byť priveľmi rozdielne.

V prvom kroku sa vyberie jeden z bodov predstavujúcich referenčné diódy. K tomuto bodu sa potom hľadá najbližší nereferenčný bod, ktorý spĺňa podmienku vzdialenosti. Po nájdení takéhoto bodu sa pokračuje hľadaním ďalšieho nereferenčného bodu. V tomto prípade sa už porovnávajú uhly priamok oboch dvojíc bodov. Postup sa opakuje do nájdenia n bodov. V prípade, že nie je možné nájsť vhodné body tvoriace n-ticu sú body vyradené z množiny a pokračuje sa ďalším referenčným bodom.

Ako pozícia robota sa volí stred úsečky medzi prvým a posledným bodom. Orientácia sa získava vzhľadom k ose X. Ide uhol, ktorý zviera priamka prechádzajúca prvým a posledným bodom a osou X, od ktorého je odpočítaných 90˚.

7.6 Testovanie aplikácie

Počas vývoja bola aplikácia viackrát testovaná. Prvotné testovanie zahrňovalo iba vyhľadávanie pozície LED diód v obraze, pričom boli použité fotografie ktoré poskytovali ideálne podmienky pre algoritmus.

Po získaní záberov, ktoré neposkytovali ideálne podmienky bola zistená prílišná závislosť algoritmu od svetelných podmienok. Toto si vyžiadalo úpravu algoritmu a zmenu farebného modelu použitého v aplikácii.

Prvotná verzia aplikácie eliminovala perspektívne skreslenie transformáciou celej časti obrazu predstavujúcu rovinu obsahujúcu LED diódy. V transformovanej rovine sa následne vyhľadávala pozícia diód. Pri testovaní bolo zistené, že takéto riešenie je príliš pomalé, keďže transformácia veľkej časti obrazu je časovo pomerne náročná operácia. Vzhľadom na túto skutočnosť boli algoritmy upravené tak, aby sa pozície diód vyhľadávani v netransformovanom (originálnom) obraze. Pozície sú potom transformované samostatne, bez nutnosti operácií nad celým obrazom.

7.7 Návrhy na ďalšie rozšírenia

Nedostatkom aplikácie je pomerne veľa manuálnych krokov, ktoré musí užívateľ vykonať pri kalibrácii aplikácie. K odstráneniu týchto krokov je, okrem úpravy aplikácie, nutná

aj modifikácia hracej plochy. Po označení okrajov ihriska by bolo možné automatické získanie ich pozície, čím by bola umožnená automatická kalibrácia kamery.

Ďalší problém, ktorý aplikácia nerieši je identifikácia lopty použitej pri hre. Pre jej vyhľadanie je možné využiť upravené verzie použitých algoritmov, pričom by bolo potrebné zmeniť niektoré parametre.

8 Záver

Táto diplomová práca vznikla ako súčasť projektu fotbalu robotov. Jedným z prínosov práce je vytvorenie služby, ktorá umožňuje ovládanie robotov. Okrem toho poskytuje rozhranie pre ostatné služby.

Hlavným prínosom práce je vyvinutie algoritmov potrebných k identifikácii viacerých robotov na hracej ploche. Pre označovanie robotov boli použité už hotové hardwarové moduly. Práca poskytuje knižnicu tried, ktorá umožňuje lokalizáciu robotov identifikovaných týmito modulmi. Software vyžaduje niekoľko manuálnych krokov, nutných k jeho správnej kalibrácií.

Je potrebné vytvoriť označenie okrajov ihriska majákmi, ktoré by aplikácia mohla spracovať. Tým by sa odstránila časť manuálnych krokov.

Ďalej je potrebné vyladenie algoritmu detekujúceho škvrny v obraze. Riešením by mohlo byť vizuálne označovania robotov, ktoré nie je založené na použití LED diód.

Výsledná aplikácia demonštruje princípy a postupy, ktoré je možné použiť k určeniu pozície, identity a otočenia viacerých robotov v rámci hracej plochy. Je nutné vytvoriť prepojenie medzi touto aplikáciou a službami poskytujúcimi ovládanie robotov.

9. Použitá literátúra

[1] *Microsoft Robotics Developer Center* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://msdn.microsoft.com/robotics/>.

[2] *LEGO Mindstorms* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://mindstorms.lego.com/>.

[3] PASTRŇÁK, Ján. Microsoft Robotics Studio - Programování robotů. VŠB - TU Ostrava, Fakulta elektrotechniky a informatiky, 2008. 47 s. Diplomová práce. VŠB - TU Ostrava.

[4] Eduard Sojka, *Digitální spracování obrazu*, VŠB-TU Ostrava, 1999.

[5] *OpenCV* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://opencv.willowgarage.com/wiki/>.

[6] Camera resectioning In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 7 August 2006, 26 Marec 2010 [cit. 2010-05-05]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Camera_resectioning>.

[7] Eduard Sojka, *Počitačová grafika II*, VŠB-TU Ostrava, 1999.

[8] *EmguCV* [online]. 2010 [cit. 2010-05-03]. Dostupné z WWW: <http://emgu.com/>.

A Obsah DVD

- Aplikacie – Aplikacia pre vyhľadávanie robotov. Implementacia DSS služby pre ovladanie robota

- Prilohy – Prílohy k diplomovej práci.

    - 1-Obsah DVD.pdf – tento obsah DVD.

    - 2-Programatorska dokumentacia sluzby.pdf – podrobná programátorská dokumentácia k DSS službe, ktorá bola uvedená v kapitole 4.

    - 3-Programatorska dokumentacia vyhladavania.pdf – podrobná programátorská dokumentácia aplikácie pre vyhľadávanie robotov.

    - 4-Navod na testovanie sluzby.pdf – návod na otestovanie DSS služby, ktorá bola uvedená v kapitole 4.

    - 5-Navod na testovanie aplikacie.pdf – návod na otestovanie aplikácie pre vyhľadávanie robotov.

- Testovacie fotografie – Súbor testovacích fotografií použitých pri vývoji aplikácie.

- Text – Text diplomovej práce vo formáte .docx a fotokópia zadania diplomovej práce.